VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

SYSTÉM PRO MONITOROVÁNÍ SÍŤOVÝCH PROTOKOLŮ

SYSTEM FOR MONITORING OF NETWORK PROTOCOLS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. ROMAN SELECKÝ**
AUTHOR

VEDOUCÍ PRÁCE — **JAN KOŘENEK, Ph.D.**
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Selecký Roman, Bc.**
Obor: Počítačové sítě a komunikace
Téma: **Systém pro monitorování síťových protokolů**
System for Monitoring of Network Protocols
Kategorie: Počítačová architektura

Pokyny:

1. Nastudujte problematiku analýzy a extrakce hlaviček paketů v technologii FPGA.
2. Seznamte se s možnostmi generování kódu z jazyka P4.
3. Navrhněte systém pro monitorování 10 Gb sítí, který umožní poskytnout informace o struktuře síťového provozu ve formě flow dat a stromu protokolů a současně umožní zachytit pakety na základě sekvence zapouzdření protokolů.
4. Vytvořte implementaci navrženého systému nad vhodnou vestavěnou platformou využívající výkonný procesor ARM a technologii FPGA.
5. Pro vytvořený systém navrhněte a implementujte vhodné webové uživatelské rozhraní.
6. Vytvořenou implementaci důkladně otestujte a změřte parametry vytvořeného vestavěného systému.
7. V závěru diskutujete dosažené výsledky a možnosti pokračování projektu.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kořenek Jan, Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Monitorovanie sietí je potrebné najmä pri diagnostike, riešení problémov, detekcii nežiadúcej prevádzky a podozrivého zapúzdrenia hlavičiek protokolov. Preto je cieľom tejto práce vytvoriť návrh a implementáciu systému pre monitorovanie 10 Gb sietí, ktorý poskytuje informácie o štruktúre sieťovej prevádzky a zároveň umožňuje zachytiť pakety na základe sekvencie zapúzdrenia protokolov. Pre dosiahnutie potrebnej priepustnosti bola implementácia paketového analyzátora a filtra paketov hardvérovo akcelerovaná v FPGA. Flexibilitu zabezpečuje využitie nástroja mapujúceho program v jazyku P4, ktorý popisuje spracovávanie paketov, do jazyka VHDL. Na základe informácií získaných pri analýze paketov sú vytvárané záznamy o tokoch a tie sú ukladané pomocou protokolu IPFIX. Prostredníctvom grafického rozhrania sú zozbierané informácie zobrazované užívateľovi vo forme stromu protokolov, ktorého uzly sú asociované so záznamami o tokoch.

Abstract

It is necessary to monitor networks namely for diagnostics, troubleshooting, detection of anomalies and suspicious header encapsulations. This thesis aims to design and implement a system for monitoring protocol structure on 10 Gb networks, which will be able to capture packets based on the sequence of encapsulated protocols. To achieve requested throughput some tasks like packet parsing and packet filtering were accelerated in FPGA. Flexibility is achieved by using a tool that maps P4 programs, which define packet parsing process, to VHDL language. Based on the information gained from packet parsing, flow records are created and stored via IPFIX protocol. This information is displayed through a graphical user interface in the form of protocol tree, whose nodes are associated with flow records.

Kľúčové slová

monitorovanie tokov, strom protokolov, filtrovanie paketov, FPGA, P4, IPFIX, 10 Gb siete

Keywords

flow monitoring, protocol tree, packet filtering, FPGA, P4, IPFIX, 10 Gb networks

Citácia

SELECKÝ, Roman. *Systém pro monitorování síťových protokolů.* Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Jan Kořenek, Ph.D.

Systém pro monitorování síťových protokolů

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána doktora Jana Kořenka. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Roman Selecký
21. mája 2018

Poďakovanie

V prvom rade, by som rád poďakoval vedúcemu diplomovej práce za všetok obetovaný čas a neoceniteľné rady. Ďalej všetkým spolupracovníkom z projektu SProbe za ich trpezlivú pomoc.

Obsah

Kapitola 1

Úvod

Výpočtová technika sa dostáva do všetkých vedných disciplín a každého odvetvia priemyslu. Rovnako enormný rast využívania technológií môžeme sledovať aj v každodennom živote ľudí. K prudkému nárastu užívateľov prispievajú aj nové služby sprostredkúvajúce užívateľom veľké množstvo rôznych informácií. Tým pádom sa rýchlejšie ako počet užívateľov zvyšuje objem prenesených dát. Na tieto zvyšujúce sa nároky je potrebné reagovať najmä zmenami v sieťovej infraštruktúre. Preto sa siete a ich správa stávajú oveľa komplexnejšími a zložitejšími, čo vedie k zvýšeniu pravdepodobnosti chýb. Keďže pri odhaľovaní chýb je prvým a najdôležitejším krokom ich izolovanie, je nutné vedieť, čo presne sa na sieti deje.

Potreba monitorovania sieťovej prevádzky vznikla spolu s počítačovými sieťami. V tom čase bolo nutné najmä spravovať a monitorovať samotné sieťové prvky. Za týmto účelom začali vznikať prvé protokoly, ktoré sledovali zariadenie zvnútra. Na základe prístupu k samotnému zariadeniu, vedeli v prípade potreby odoslať hlásenie o zmene stavu alebo chybe. Taktiež umožňovali vzdialene pristupovať a meniť interné nastavenia na zariadení. Príkladom takéhoto protokolu je SNMP.

Postupom času sa povaha sietí výrazne zmenila. Z množstva izolovaných sietí, ktoré boli spravované jednou organizáciou, vznikol Internet, prepájajúci obrovské množstvo zariadení, ktoré patria jednotlivcom. Zvyšovanie počtu užívateľov a sieťových služieb, teda jasne ukázalo, že metóda monitorovania zariadení nestačí, pretože vyžaduje prístup do zariadenia. Taktiež sa zmenili požiadavky monitorovania a viac ako stav konkrétnych zariadení sa začala sledovať komunikácia, ktorá sieťou prechádza. Aby sme mohli získané informácie efektívne spracovávať a interpretovať, bol zavedený pojem tok, angl. *flow*, ktorý agreguje pakety do logických skupín [4]. Na základe tejto klasifikácie vznikli viaceré protokoly, najmä NetFlow a IPFIX.

Monitorovanie tokov síce informuje o priebehu komunikácie dvoch zariadení, ale neobsahuje informácie o jej štruktúre, ktorú vyjadruje práve nadväznosť hlavičiek protokolov v paketoch. Tieto informácie sú potrebné pri správe sietí, hlavne v kombinácií s monitorovaním tokov. Preto bol v rámci práce navrhnutý nový spôsob monitorovania sietí, ktorý agreguje postupnosti hlavičiek vo forme acyklického neorientovaného grafu, tzv. *"stromu protokolov"*.

S rýchlosťou liniek rádovo v 10 Gbps, narážajú konvenčné metódy, spracovávajúce veľké množstvo dát, na výkonnostné limity. Keďže softvérové riešenia nie sú schopné dosiahnuť požadovanú priepustnosť, používa sa hardvérová akcelerácia časovo kritických operácií. Medzi ne patrí najmä spracovávanie paketov za cieľom získania informácií potrebných pri monitorovaní sieťovej prevádzky.

Na dosiahnutie potrebnej priepustnosti je možné využiť hardvérovú akceleráciu v FPGA. Prínosom tohto riešenia je softvérovo definované spracovávanie protokolov popísané v jazyku P4. Spojenie vysoko-úrovňového popisu s možnosťou rekonfigurácie technológie FPGA vedie ku flexibilite, ktorá je pri súčasnom trende vzniku nových protokolov veľmi potrebná.

Cieľom tejto práce je navrhnúť a implementovať monitorovací systém určený k nasadeniu v 10 Gb sieťach. Jeho úlohou je vykonávať monitoring tokov v kombinácií s monitorovaním štruktúry protokolov tak, aby jednotlivé záznamy o tokoch boli asociované s odpovedajúcim uzlom stromu protokolov. Tento systém musí navyše podporovať zachytávanie paketov na základe ich štruktúry. Všetky výstupné dáta je nutné natrvalo ukladať a prostredníctvom užívateľského rozhrania zobrazovať.

V kapitole 2 sú predstavené a porovnané základné prístupy monitorovania sietí, ako aj vývoj zástupcov týchto prístupov. Kapitola 3 sa zameriava na spracovanie sieťovej prevádzky, spôsob reprezentácie štruktúry protokolov a možnosti využitia jazyka P4 pri týchto úlohách. Kapitola 4 sa venuje mapovaniu jazyka P4 do VHDL a obsahuje popis fungovania jednotky určenej k analýze paketov. Charakteristika cieľovej platformy, pre ktorú je systém monitorovania sieťových protokolov navrhovaný, sa nachádza v kapitole číslo 5. Kapitola 6 obsahuje popis mapovania úloh na výpočtové časti monitorovacieho systému, návrh architektúry jednotlivých častí a popis ich implementácie. Logická štruktúra grafikého rozhrania a spôsob interakcie s užívateľom je opísaný v kapitole 7. Kapitola 8 popisuje testovacie prostredia, ktoré boli implementované za účelom overenia správnej funkcionality systému. Výsledky meraní parametrov monitorovacieho systému sú zahrnuté v kapitole 9.

Kapitola 2

Monitorovanie sieťovej prevádzky

Potreba monitorovania sieťovej prevádzky vznikla spolu s počítačovými sieťami. Spolu s vyvíjajúcimi sa sieťovými technológiami, sa menili aj metódy monitorovania. Cieľ, ktorým je zabezpečenie dostupnosti, však ostáva rovnaký. Okrem toho monitorovanie sietí poskytuje dôležité informácie pri identifikácii, diagnostike a riešení problémov, plánovaní zmien, účtovaní a zaisťovaní bezpečnosti.

2.1 Metódy monitoringu

Sieťový monitoring, ako súčasť sieťového manažmentu zodpovedá za získavanie potrebných informácií o intenzite a charaktere dát prenášaných po sieti. Na základe typu informácií, stupňa agregácie metadát a doby spracovania rozlišujeme tri základné metódy monitoringu.

- **Monitorovanie sieťových zariadení** je zamerané na sledovanie stavu aktívnych sieťových prvkov. Vo všeobecnosti poskytuje možnosť čítať a zapisovať hodnoty zo zariadení. Jedná sa o agregované, jednorozmerné metadáta napr. počet prijatých paketov na sieťovom rozhraní alebo teplota procesoru. Zástupcom tohoto prístupu je SNMP.
- **Monitorovanie sieťových tokov** vychádza z rozdelenia sieťovej prevádzky na toky, ktoré sú podľa [14] definované ako množina IP paketov prechádzajúcich monitorovaným bodom v sieti počas určitého časového intervalu. Všetky pakety, patriace do jedného toku, angl. *flow*, musia mať rovnaké vlastnosti. Vlastnosť je definovaná ako funkcia hodnoty poľa z hlavičky protokolu alebo charakteristika samotného paketu, napr. zdrojová IP adresa alebo počet MPLS návestí. Pri monitorovaní dátových tokov sú všetky pakety klasifikované, podľa vybraných vlastností, do tokov. Používateľovi sú poskytované sumarizované informácie o jednotlivých tokoch. Zástupcami tohto typu monitorovania sú IPFIX a NetFlow.
- **Hĺbková analýza paketov** poskytuje užívateľom najväčšie množstvo informácií o prevádzke na sieti. Tento prístup neumožňuje agregáciu, pretože uchováva informácie o každom spracovanom pakete samostatne. Nad uloženými dátami je možné neskôr vykonávať široké spektrum analýz, ktorých časová a priestorová náročnosť rastie s počtom skúmaných paketov. Zástupcom tohoto prístupu je analyzátor Wireshark.

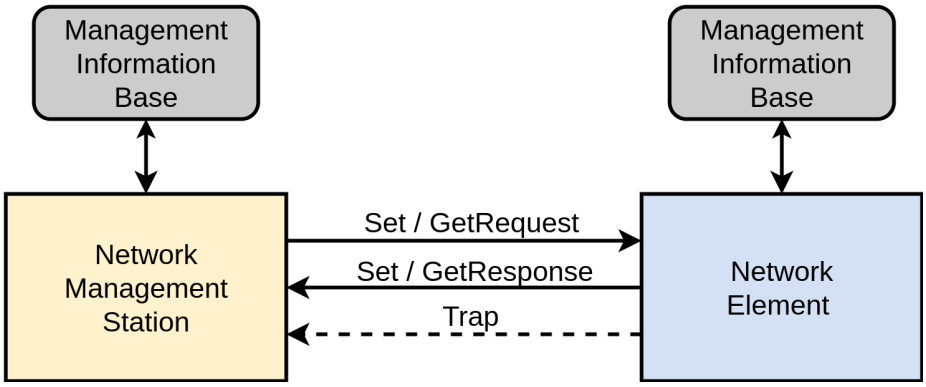
2.2 SNMP

Ako z názvu *Simple Network Management Protocol* vyplýva, SNMP je základným protokolom určeným k správe sieťových zariadení. Umožňuje zbierať a štruktúrovane uchovávať informácie o spravovaných zariadeniach. Navyše špecifikuje spôsob, akým pozmeňovať konfiguráciu a tým meniť chovanie sieťových prvkov.

Tento protokol bol štandardizovaný v roku 1988 organizáciou **IETF**, z angl. *Internet Engineering Task Force*. Prvá skupina dokumentov RFC, z angl. *Request For Comments*, číslo 1065, 1066 a 1067, popisuje základnú štruktúru a identifikačnú schému potrebnú pri definovaní riadiacich informácií, využívaných pri správe zariadení v TCP/IP sieťach. Ďalej obsahuje popis informačného modelu objektov a ich typov [29]. Posledný dokument z tejto série popisuje architektúru a spôsob komunikácie medzi SNMP entitami [5].

2.2.1 Architektúra SNMP

Model architektúry SNMP pozostáva z monitorovacích staníc, angl. *network management stations* a sieťových prvkov, angl. *network elements*. Monitorovacia stanica, označovaná tiež ako SNMP Manager spúšťa riadiace aplikácie, ktoré dohliadajú na sieťové prvky. Medzi sieťové prvky patria napríklad klientské stanice, servery, smerovače alebo prepínače, obsahujúce SNMP Agentov, ktorí vykonávajú monitorovacie funkcie vyžiadané SNMP Managerom [5]. Transportným protokolom využívaným pri komunikácii SNMP entít je UDP. Celá komunikácia prebieha formou odosielania správ, kde každá správa je prenášaná samostatne, v jedinom UDP datagrame. Na obrázku 2.1 môžeme vidieť spôsob komunikácie medzi monitorovacou stanicou a sieťovým prvkom. Správy, ktoré si zariadenia vymieňajú, rozdeľujeme podľa typu komunikácie do dvoch skupín:

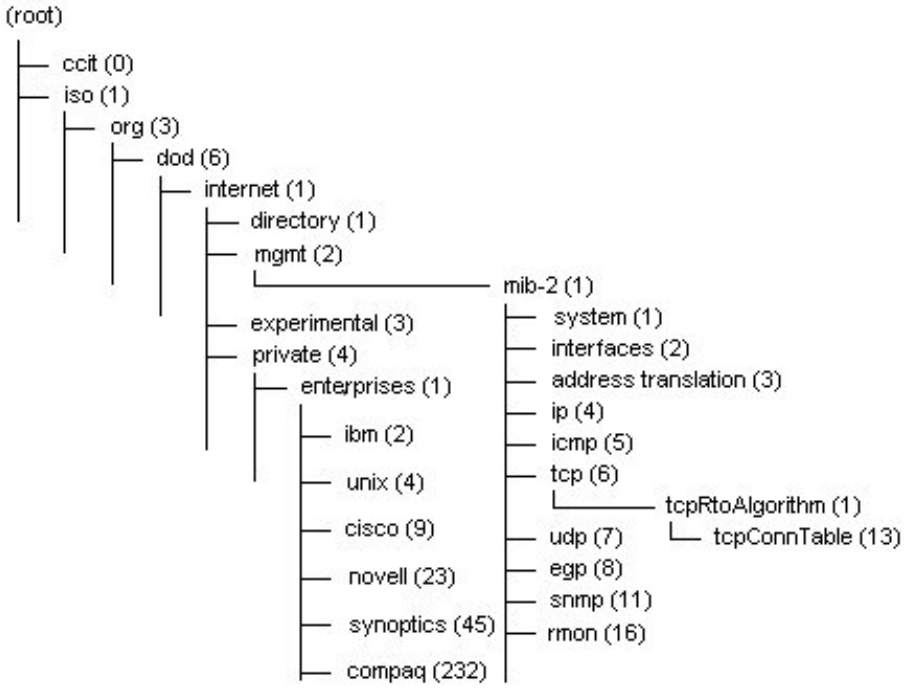


Obr. 2.1: Schéma komunikácie prostredníctvom protokolu SNMP

- **Synchrónna komunikácia** sa využíva pri výmene väčšiny správ. Monitorovacia stanica vždy začína komunikáciu správou typu: **SetRequest** alebo **GetRequest**. Sieťový prvok odpovedá správou typu **SetResponse** alebo **GetResponse**, ktorá obsahuje informácie požadované SNMP Managerom. Synchrónna komunikácia je na obrázku 2.1 zobrazená plnou čiarou. SNMP aplikácie komunikujú synchrónne na porte 161.
- **Asynchrónna komunikácia** je využívaná v prípade, že SNMP Agent zaznamená dôležitú udalosť na sieťovom prvku, o ktorej informuje nevyžiadanou správou monitorovaciu stanicu. Táto správa sa v rámci SNMP nazýva **Trap** a obsahuje identifikátor

udalosti, ktorá zapríčinila odoslanie tejto správy. Asynchrónna komunikácia je na obrázku 2.1 zobrazená prerušovanou čiarou a je možné vidieť, že SNMP Manager neodpovedá žiadnou ďalšou správou. SNMP aplikácie komunikujú asynchrónne na porte 162.

Aby mohol byť protokol SNMP podporovaný výrobcami všetkých sieťových zariadení, bol definovaný hierarchicky usporiadaný menný priestor popisujúci objekty, ku ktorým je možné pristupovať. Ide o abstraktnú štruktúru typu strom, ktorá je nazývaná MIB, z angl. *management information base*. Na obrázku 2.2 môžeme vidieť, že každý uzol stromu je identifikovaný číslom, ktoré je v rámci danej úrovne unikátne. Objekty sú v MIB identifikované na základe mien vo forme postupnosti čísel tzv. **OID**, ktorá reprezentuje cestu v strome [30].



Obr. 2.2: Vybrané uzly MIB určené k správe zariadení v TCP-IP sieťach [19]

Prvá verzia protokolu, SNMPv1, bola kritizovaná najmä kvôli slabej úrovni zabezpečenia. Autentizácia agentov voči monitorovacej stanici využívala tzv. *"community string"*, teda heslo, ktoré je prenášané po sieti v otvorenej podobe. Skutočnosť, že správy neboli šifrované alebo inak zabezpečené umožňovala rôzne typy útokov napr. podvrhnutie alebo pozmenenie správ.

Ďalšia verzia, SNMPv2, priniesla zmeny nielen v oblasti bezpečnosti. Navrhovaný bezpečnostný model obsahoval okrem iného aj metódy zabezpečujúce **autenticitu** a **integritu** [36]. Taktiež bol pridaný nový typ synchrónnej správy: **GetBulkRequest**, ktorá je určená k vyžiadaniu veľkého množstva dát jednou správou. Za účelom škálovateľnosti bol pridaný typ správy: **InformRequest**. Slúži ku komunikácii medzi SNMP Managermi a umožňuje vytváranie hierarchie monitorovacích staníc. SNMPv2 nebol dobre prijatý, čo bolo zapríčinené jeho komplikovanosťou. Keďže sa nepodarilo štandardizovať navrhované zmeny v oblasti bezpečnosti, SNMPv2 využíva staršiu a slabšiu variantu zabezpečenia zo SNMPv1. To viedlo k vzniku jeho ďalších verzií: SNMPv2c, SNMPv2u a SNMPv2*.

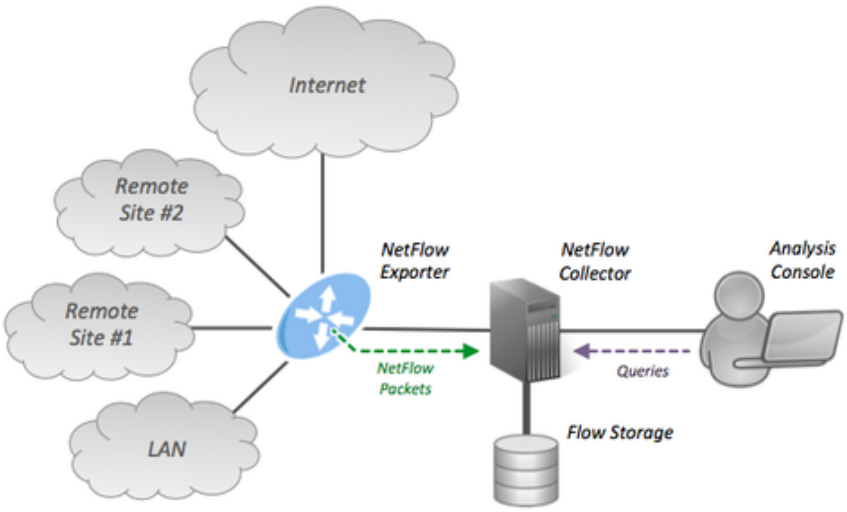
Posledným štandardom z rodiny SNMP je SNMPv3, ktorý prináša zmeny hlavne v bezpečnosti. Využíva kryptografické algoritmy a tým zabezpečuje **integritu**, **dôvernosť** a **autentizáciu**. Vďaka tomu je možné bezpečne využívať SNMP aj k vzdialenej konfigurácii sieťových prvkov.

2.3 NetFlow

Protokoly na správu sieťových prvkov, ako SNMP, nám umožňujú zisťovať napr. záťaž CPU na zariadení, využitie pamäti alebo stav a vyťaženie jednotlivých sieťových rozhraní. Prostredníctvom týchto protokolov nedokážeme zistiť to, aká prevádzka sieťou prechádza. Vďaka **monitorovaniu sieťových tokov** získavame kvalitatívne orientovaný pohľad [23], preto ako odpoveď na potrebu monitorovania sieťovej prevádzky za účelom diagnostiky, detekcie anomálií a validácie parametrov kvality služieb [8], vznikol **NetFlow**. Tento protokol je od roku 1996 [15] integrovaným do veľkej časti produktov spoločnosti **Cisco Systems, Inc.** Odvtedy sa NetFlow stal synonymom monitorovania sieťových tokov, nakoľko sa však jedná o proprietárny protokol, mnohé ďalšie spoločnosti vyvinuli vlastné riešenia, napr. J-Flow, Rflow, Netstream a iné.

2.3.1 Architektúra NetFlow

Základná architektúra sa skladá z tzv. *"NetFlow exportéra"* a *"Netflow kolektoru"*. Exportérom môže byť sieťové zariadenie vykonávajúce svoju primárnu funkciu, napr. smerovač, alebo sa môže jednať o dedikované zariadenie pripojené k sledovanej linke pomocou rozbočovača. Kolektorom nazývame softvér alebo počítač, ktorý prijíma záznamy o tokoch odosielané exportérom. Úlohou kolektoru je ukladať tieto záznamy na úložisko a sprístupňovať ich užívateľom za účelom ďalších analýz.



Obr. 2.3: Architektúra NetFlow v sieti [25]

Komunikácia medzi exportérom a kolektorom je vždy jednosmerná a asynchrónna. Ako môžeme na obrázku 2.3 vidieť, `NetFlow exporter` odosiela podľa potreby pakety obsahujúce NetFlow záznamy na `NetFlow Collector`. Pri tejto komunikácií NetFlow protokol

neobsahuje mechanizmus, ktorý by slúžil na potvrdzovanie prijatých záznamov alebo odosielanie žiadostí z kolektoru na exportér. Naopak, komunikácia medzi užívateľom a kolektorom je synchrónna. Užívateľ vytvára žiadosti a na základe dopytu, kolektor vo svojom úložisku vyhľadá odpovedajúce dáta a odosiela ich užívateľovi.

NetFlow exportér postupne vytvára záznamy o tokoch, do ktorých patria pakety prichádzajúce na jeho sieťové rozhrania. Okrem položiek získaných pri analýzach paketov, ako zdrojová a cieľová IPv4 adresa, zdrojový a cieľový L4 port a L3 protokol, obsahujú záznamy o tokoch agregované informácie napr. počet paketov v toku, veľkosť toku v bajtoch a časové značky začiatku a konca toku. Za začiatok toku je považovaný čas príchodu prvého paketu. Situácia ohľadom konca toku je komplikovanejšia, pretože závisí na type použitého transportného protokolu. Spojovo-orientované protokoly, ako napr. TCP, obsahujú mechanizmus, ktorým si komunikujúce strany oznamujú, že ukončujú komunikáciu. Nespojované protokoly, ako napr. UDP, obdobný mechanizmus nepodporujú, preto je komunikácia považovaná za ukončenú, keď si účastníci prestanú vymieňať datagramy. Pri monitorovaní tokov, môže byť záznam o toku uzavretý v prípade jednej z týchto udalostí [39] :

- **Neaktívny časovač** ukončuje zbieranie štatistík a zahájí export odpovedajúceho záznamu v prípade, že za posledných 15 sekúnd nebol prijatý žiadny paket patriaci do tohoto toku.
- **Aktívny časovač** uzatvára záznamy o toku trvajúce viac než 30 minút. Tým pádom môže nastať situácia, že jeden dlhotrvajúci tok, napr. prenos súborov pomocou FTP, bude rozdelený do viacerých záznamov.
- **Koniec toku**, ktorý na transportnej vrstve využíva spojovo-orientovaný protokol. K uzavretiu záznamu o toku dochádza na exportéri pri ukončení spojenia jednou z komunikujúcich strán. V prípade TCP, sa spojenie uzatvára správou s príznakom FIN alebo RST.
- **Obmedzenie na exportéri**, ako napr. nedostatok pamäte, môže vynútiť uzavretie záznamov o tokoch a ich následný export na kolektor.

Vyššie uvedené hodnoty aktívneho a neaktívneho časovača, angl. *time-out*, sú východiskové. V rámci konfigurácie ich môžeme zmeniť a tým prispôsobiť politiku monitorovania tokov charakteru prevádzky v konkrétnej sieti [8].

Od prvej verzie protokolu NetFlow, export paketov prebiehal v tzv. *"dávkach"*, čo znamená že po ukončení zberu štatistík toku, nebol odpovedajúci záznam o toku okamžite poslaný po sieti na kolektor. Za účelom minimalizácie réžie, ktorú so sebou prináša odosielanie, prenos a prijímanie správ, sú viaceré záznamy združené v jednom pakete.

Vo verzii 1 bol na prenos záznamov o tokoch využívaný protokol UDP. Z jednoduchosti tohto transportného protokolu vyplýva tiež niekoľko nevýhod. Keďže NetFlow neobsahuje mechanizmus potvrdzovania prijatých flow záznamov, pri používaní UDP môže dôjsť ku nedetekovateľným stratám paketov obsahujúcich množstvo záznamov. Ďalším problémom je viacnásobné prijatie jedného paketu, čo môže spôsobiť duplikácie tokov a tým skreslenie analýz zozbieraných dát. Taktiež v prípade poruchy kolektoru, sa exportéry o jeho výpadku nedozvedia a nebudú môcť na vzniknutú situáciu reagovať, napr. posielaním dát na sekundárny kolektor.

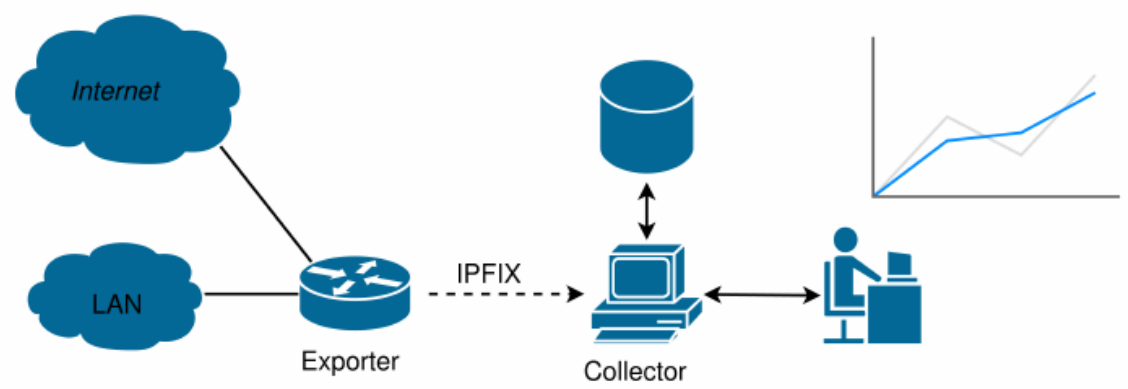
Vo verzii 5 bola pridaná podpora prenosu pomocou TCP. Keďže aj prenos cez TCP má svoje nevýhody, od verzie 5 obsahuje hlavička protokolu NetFlow sekvenčné čísla. Avšak stále je možné, v rámci záznamov o tokoch, prenášať len niektoré, preddefinované údaje.

S ďalším rozvojom sietí a vzniku nových protokolov vznikol NetFlow verzie 9, ktorého najdôležitejším vylepšením oproti verzii 5, je podpora šablón. Šablóny umožňujú dynamické vytváranie štruktúry záznamov o tokoch. Užívateľ si môže nadefinovať vlastné polia, tým pádom aj doteraz nepodporované IPv6 adresy, VLAN a MPLS tagy. Vďaka šablónam je taktiež možné jedným exportérom vytvárať niekoľko rôznych typov záznamov. Navyše veľkosť záznamov nemusí byť fixná, teda nevyužité polia nemusia byť prenášané a tak môže dôjsť k zníženiu objemu dát posielaných na kolektor [9].

2.4 IPFIX

IPFIX je protokol štanadardizovaný skupinou IETF. Ako z názvu *Internet Protocol Flow Information Export* vyplýva, jedná sa o otvorený a univerzálny štandard, ktorý definuje spôsob, akým sú záznamy o tokoch formátované, prenášané a ukladané [10]. Štandard IPFIX je založený na dokumente RFC 3954 [9] popisujúcom protokol NetFlow verzie 9, preto býva niekedy označovaný ako NetFlow verzie 10.

Z hľadiska architektúry , IPFIX neprináša žiadne výrazné zmeny. Ako môžeme na obrázku 2.4 vidieť, role a spôsob vzájomnej komunikácie medzi časťami monitorovacieho systému sa nezmenili. Tiež si môžeme všimnúť, že používaná terminológia prešla oproti protkolu NetFlow minimálnymi zmenami.



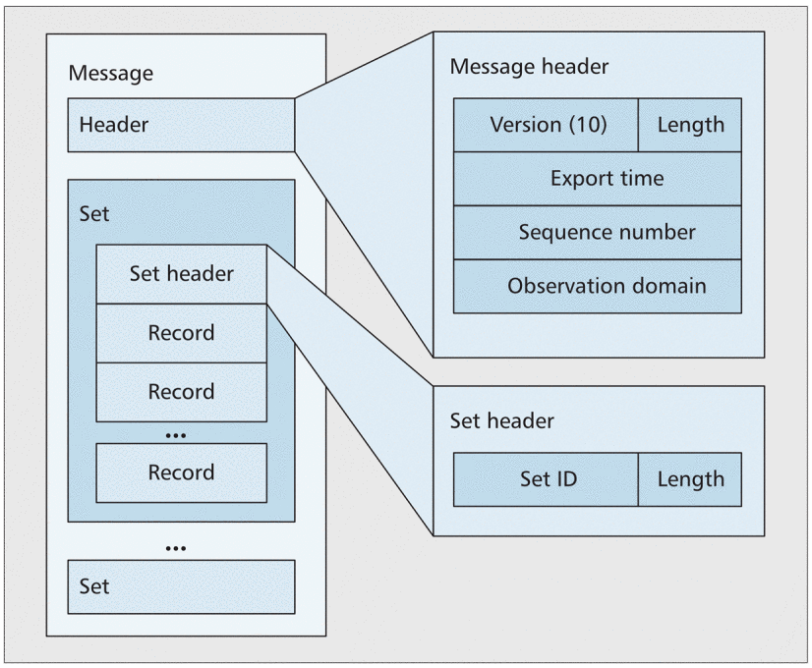
Obr. 2.4: Schéma komunikácie prostredníctvom protokolu IPFIX

Napriek mnohým spoločným vlastnostiam IPFIX prekonáva a nahrádza NetFlow. V porovnaní s protokolom NetFlow, dáva tento otvorený štandard užívateľom oveľa viac možností [21]. Medzi najzásadnejšie zmeny patrí:

- **Variabilná dĺžka polí** v zázname o toku umožňuje maximalizovať využitie prenosových kapacít medzi exportérom a kolektorom. Taktiež je možné prenášať detailnejšie informácie, ktorých veľkosť vopred nepoznáme. Medzi takéto položky patrí napr. URL webovej stránky prehľadávanej protokolom HTTP.
- **Identifikátor výrobcu**, z angl. *Vendor ID* alebo tiež *Private Enterprise Number*, umožňuje výrobcom IPFIX zariadení definovať štruktúru exportovaných záznamov obsahujúcu proprietárne údaje a stále spĺňať štandard. Takže je možné zužitkovať výhody protokolu IPFIX a používať ho aj na prenos a uchovávanie SNMP správ.

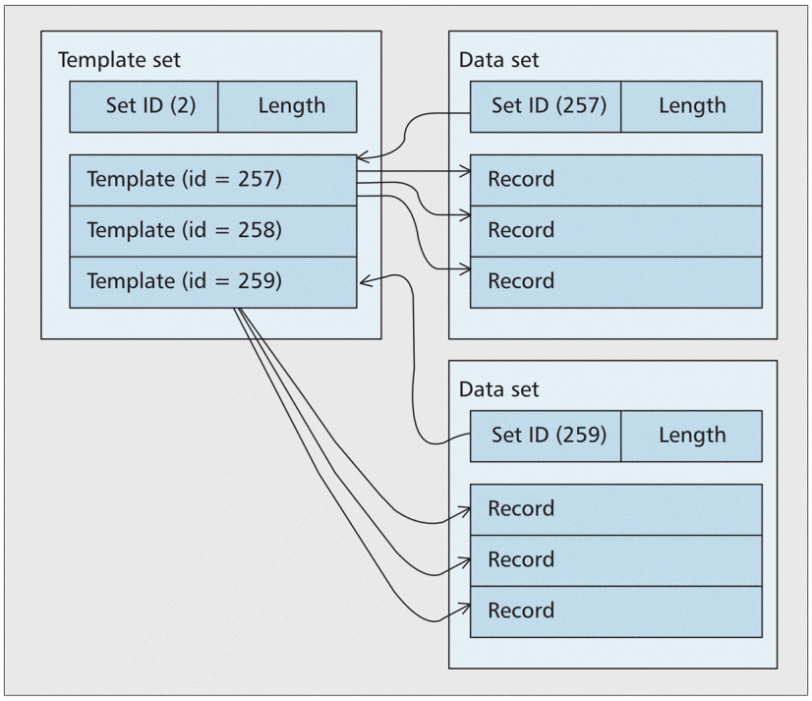
Základnou jednotkou prenosu je v protokole IPFIX správa, angl. *message*. Každá správa pozostáva z hlavičky, angl. *header* a aspoň jednej skupiny, angl. *set*. Existujú dva druhy skupín. Prvou je dátová skupina, angl. *data set*, v ktorej sa nachádzajú záznamy o tokoch. Druhou je šablónová skupina, angl. *template set*, obsahujúca záznamy definujúce jednotlivé šablóny, ktoré špecifikujú formát dátových záznamov. Na obrázku 2.5 je znázornená IPFIX správa, ktorá obsahuje nasledujúce položky [11]:

- **Číslo verzie**, angl. *Version*, definuje verziu štandardu IPFIX, ktorej správa odpovedá. Aktuálna verzia je reprezentovaná hodnotu `0x000a`.
- **Dĺžka**, angl. *Length*, vyjadruje dĺžku celej správy v bajtoch.
- **Čas exportu**, angl. *Export time*, odpovedá UNIX-ovej časovej značke, vytvorenej pri odoslaní správy z exportéra.
- **Sekvenčné číslo**, angl. *Sequence number*, je počítadlo exportovaných dátových záznamov. Táto hodnota môže by využitá kolektorom, na detekciu straty dátových záznamov.
- **Doména**, angl. *Observation domain*, je lokálne unikátny identifikátor zariadenia, z ktorého bola správa odoslaná.



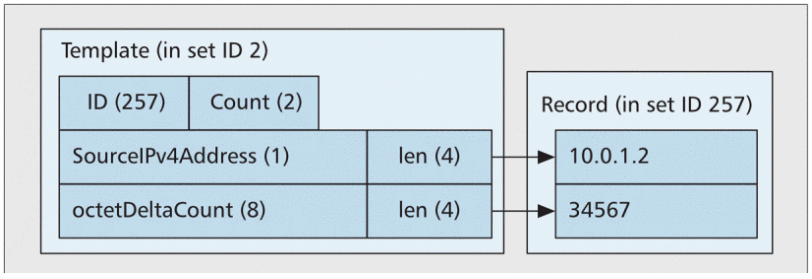
Obr. 2.5: Štruktúra IPFIX správy [35]

Rovnako ako správy, aj všetky skupiny začínajú hlavičkou. Hlavička skupiny pozostáva zo skupinového identifikátoru, angl. *Set ID* a dĺžky. Ak má identifikátor hodnotu 2, tak sa jedná o šablónovú skupinu. Každá šablóna v šablónovej skupine má svoj identifikátor šablóny, angl. *Template ID*. Skupinové identifikátory s hodnotami od 256 do 65535 sú využívané dátovými skupinami. V tomto prípade skupinový identifikátor odkazuje na šablónový identifikátor [11]. Takto sú záznamy z dátovej skupiny previazané so šablónou, ktorá definuje ich štruktúru, čo môžeme vidieť na obrázku 2.6.



Obr. 2.6: Dátová skupina a šablóna sú previazané pomocou ich identifikátorov [35].

Šablóny sú zoradené zoznamy informačných polí, angl. *information element*. Tieto polia obsahujú údaje o dátovom type a dĺžke dátovej položky. Okrem základných, sú v štandarde IPFIX definované dátové typy určené napr. k popisu adries a časových značiek. Zoznam dátových typov je spravovaný organizáciou IANA, z anglického *Internet Assigned Numbers Authority* [18], pričom každý dátový typ je identifikovaný pomocou unikátnej hodnoty. Obrázok 2.7 znázorňuje jednotlivé položky šablóny a štruktúru dátového záznamu, ktorá im odpovedá. Okrem identifikátoru šablóny, šablóna obsahuje informáciu o počte informačných polí, *Count*. Za nimi nasledujú informačné polia.



Obr. 2.7: Väzba medzi informačnými poľami šablóny a štruktúrou záznamu o toku [35]

2.5 Wireshark

Wireshark je jedným z najpoužívanejších nástrojov na hĺbkovú analýzu paketov, ktorá poskytuje užívateľom najdetailnejší pohľad na sieťovú prevádzku. Prvá verzia s názvom Et-

hereal bola vydaná pod licenciou GNU GPL už v roku 1998. Po problémoch s ochrannou značkou Ethereal, bol nástroj v roku 2006 premenovaný na Wireshark.

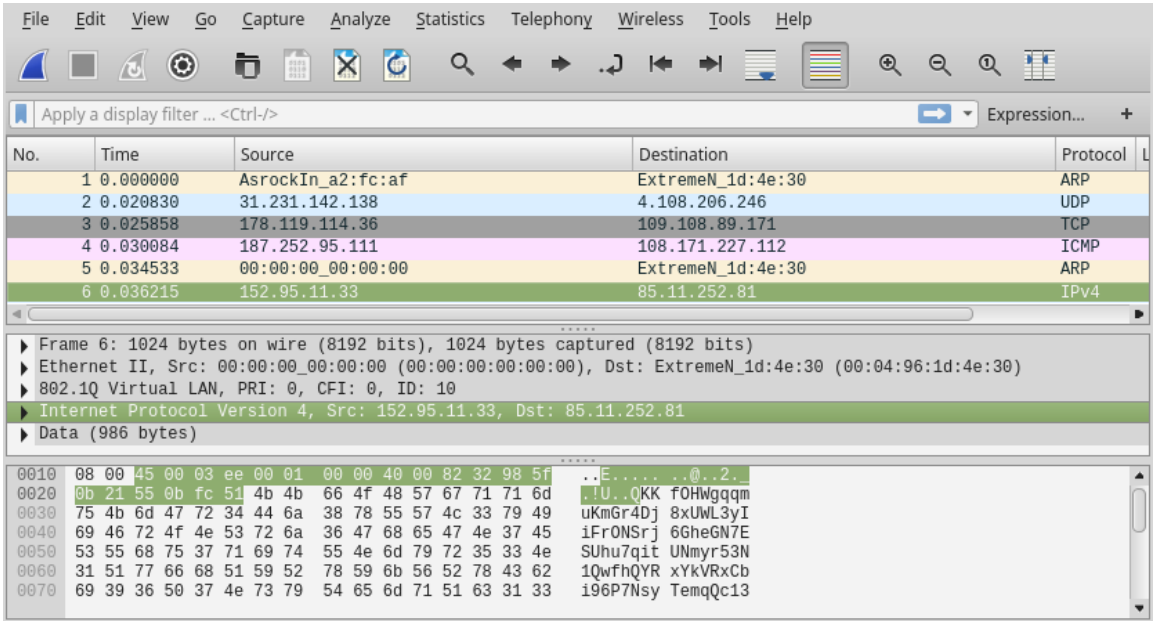
Táto aplikácia umožňuje čítať, dekódovať, zrozumiteľne zobrazovať a ukladať pakety zo siete. Wireshark môže prepnúť sieťovú kartu do promiskuitného módu, kedy má prístup ku všetkým paketom na médiu, alebo do nepromiskuitného módu, kedy sieťová karta prijíma iba pakety určené danému zariadeniu. Taktiež umožňuje offline spracovanie paketov uložených v súboroch typu **pcap**. Ďalej je veľmi dôležitá masívna podpora rozpoznávaných protokolov, z ktorých Wireshark podporuje až 750 [26]. Tento open-source program, tzn. *"s otvoreným zdrojovým kódom"*, má veľkú komunitu užívateľov a vývojárov.

Súčasťou nástroja Wireshark je množstvo analyzačných a podporných nástrojov [31]. Ich vstupom sú spracované pakety, v ktorých boli rozpoznané jednotlivé protokoly a všetky položky z ich hlavičiek. Medzi tieto nástroje patria:

- **Filtre** umožňujú pomocou výrazov presne určiť, ktoré pakety budú zahrnuté alebo vylúčené. Wireshark podporuje filtre zachytávania, angl. *capture filter* a filtre zobrazovania, angl. *display filter*.
- **Analýza koncových bodov** obsahuje základné štatistické informácie o koncových zariadeniach, angl. *endpoint*. Táto analýza sa používa najmä pri lokalizácií problému do určitého bodu v sieti.
- **Analýza sieťových konverzácií** zobrazuje užívateľovi štatistiky o obojsmerných tokoch, t.j. konverzáciách medzi dvoma koncovými zariadeniami. Medzi zobrazované informácie patria adresy zariadení, počty paketov a bajtov prenesených v oboch smeroch.
- **Analýza veľkosti paketov** poskytuje informácie o rozložení dĺžky paketov. Tento typ analýzy sa využíva pri popise charakteru prevádzky na sieti a detekcii anomálií.
- **Analýza hierarchie** informuje, akým spôsobom sú zapúzdrované hlavičky protokolov v paketoch a aké je ich percentuálne zastúpenie. Tieto informácie môžu slúžiť na detekciu útokov, ktoré zvyšujú využitie konkrétneho protokolu, napr. ARP flooding.
- **Sledovanie TCP tokov** spája celú konverzáciu, teda toky v oboch smeroch medzi koncovými zariadeniami. Dáta, takto spojenej konverzácie, je možné analyzovať ako celok, napr. súbory prenášané pomocou FTP alebo HTTP.
- **Grafy** sú častým výstupom analýz programu Wireshark. Medzi základné typy grafov patria: vstupno-výstupné, zobrazujúce priepustnosť dát v čase. Ďalej, grafy času obojsmerného prenosu znázorňujúce rozloženie RTT, z angl. *round trip time* a nakoniec grafy toku, ktoré poskytujú pohľad na konverzáciu a správy, z ktorých sa skladá, tak ako v čase nasledovali.
- **Expertné informácie** sú upozornenia generované pri analýzach paketov. Podľa závažnosti sa delia do niekoľkých kategórií a upozorňujú používateľa na neobvyklé pakety alebo chyby v hlavičkách protokolov.

Všetky tieto nástroje sú integrované v rámci grafického užívateľského rozhrania, ktoré je horizontálne rozdelené na štyri časti. Môžeme ich vidieť na obrázku 2.8, pričom nasledujúci popis postupuje zhora nadol. Prvá časť obsahuje menu a akčné tlačidlá, ktoré spúšťajú

jednotlivé nástroje. Druhú časť tvorí okno obsahujúce zoznam paketov s poradovým číslom, časom príchodu, zdrojovou a cieľovou adresou, posledným rozpoznaným protokolom a veľkosťou paketu. Po vybratí paketu, ktorý chceme bližšie skúmať sa v tretej časti zobrazí zoznam rozpoznaných protokolov s ich položkami, vyplnenými podľa obsahu paketu. Štvrtá časť zobrazuje obsah paketu po bajtoch. Zvýraznená postupnosť bajtov na obrázku zodpovedá vybratej IPv4 hlavičke v tretej časti užívateľského rozhrania.



Obr. 2.8: Grafické užívateľské rozhranie programu Wireshark

Na záver tejto kapitoly je potrebné konštatovať, že žiadny z vyššie uvedených prístupov neposkytuje informácie o štruktúre a náväznosti hlavičiek protokolov v paketoch, ktoré prechádzajú bodom vo vysokorýchlostnej sieti. Program Wireshark síce obsahuje vhodné nástroje, ale ide o offline analýzy, ktoré svojou výpočtovou a pamäťovou náročnosťou limitujú možnosti využitia iba na dočasné nasadenie. Monitorovanie sieťových tokov naopak neposkytuje požadovanú informáciu o štruktúre paketov. Agregácia do tokov však vyhovuje naším požiadavkám, pretože podľa definície [14], majú pakety jedného toku rovnaké zapúzdrenie hlavičiek protokolov. Potrebujeme preto spojiť klasifikáciu a sumarizáciu informácií o prevádzke, odpovedajúcu monitorovaniu tokov a identifikáciu štruktúry každého paketu z hĺbkovej analýzy.

Kapitola 3

Spracovanie sieťovej prevádzky s využitím jazyka P4

Na základe požiadaviek, ktoré definujú funkcionalitu monitorovacieho systému určíme, aké informácie zo sieťovej prevádzky potrebujeme získavať, tak aby sme boli schopní dané požiadavky splniť. Vhodný prístup ku získavaniu týchto informácii volíme na základe spôsobu, akým hlavičky protokolov v paketoch na seba naväzujú.

Z kapitoly 2 vyplýva, že potrebujeme spojiť klasifikáciu odpovedajúcu monitorovaniu tokov a identifikáciu štruktúry z hĺbkovej analýzy. Preto je nutné získavať informácie o sieťových tokoch a zapúzdrení hlavičiek protokolov. Vzhľadom na požadovanú flexibilitu celého systému, je vhodné pri implementácii jednotky vykonávajúcej analýzy paketov vychádzať z vysoko-úrovňového popisu náväznosti hlavičiek protokolov.

Táto kapitola popisuje problematiku analýzy paketov, jej náväznosť na získavanie informácií o štruktúre protokolov v sieťovej prevádzke a charakteristiku jazyka P4 v súvislosti s diskutovanými operáciami.

3.1 Analýza hlavičiek protokolov

Základnou vlastnosťou všetkých zariadení, ktoré chcú na sieti komunikovať, je ich schopnosť porozumieť formátu, akým sú dáta prenášané. V kontexte komunikácie na IP sieti, hovoríme o zapúzdrení protokolov. Aby zariadenie bolo schopné porozumieť prenášaným informáciám musí implementovať inverznú operáciu k zapúzdreniu. Prvým krokom pri spracovávaní hlavičiek je identifikácia protokolu. Následne môžeme, na základe popisu štruktúry hlavičky, určiť pozíciu jednotlivých polí. Nakoniec, keďže poznáme polohu potrebných dát, vykonáme ich extrakciu. Proces identifikovania a extrahovania položiek z hlavičiek protokolov sa nazýva parsovanie [32]. Okrem týchto operácií, musí byť jednotka zodpovedná za parsovanie paketov, schopná riešiť nasledujúce úlohy [13]:

- **Sekvenčná závislosť protokolov** vyplýva z definície vrstvového modelu, podľa ktorej sa protokoly zapúzdrujú postupne po vrstvách. V hlavičke sa väčšinou nachádza pole, ktoré identifikuje ďalší protokol, čo podporuje sekvenčné chovanie parserov.
- **Nejednoznačnosť nasledujúceho protokolu** sa vyskytuje napr. pri protokole MPLS. V tom prípade, hlavičky neobsahujú pole, ktoré by identifikovalo nasledujúci protokol. Postupujeme špekulatívne, napr. metódou lookahead [28], kedy sa na základe obsahu nasledujúcej hlavičky pokúšame určiť, o ktorý z možných protokolov sa jedná.

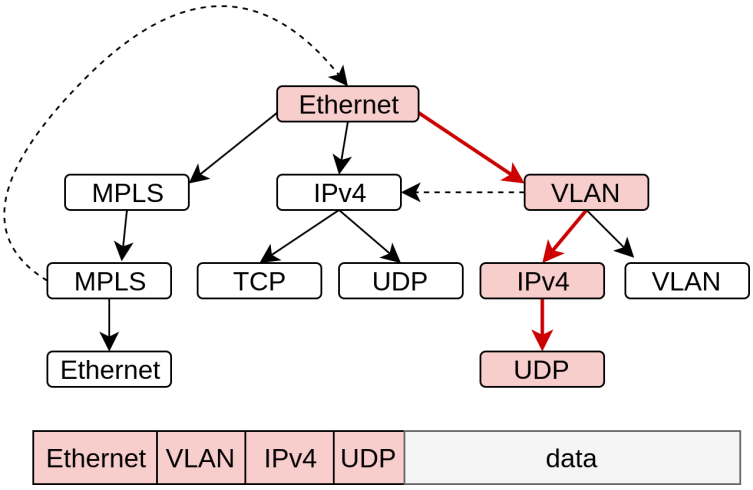
- **Premenlivá dĺžka polí** je závislá na obsahu položiek v hlavičke paketu alebo na maximálnej dĺžke stanovenej v definícií protokolu [20]. Preto táto hodnota nemôže byť vypočítaná dopredu.

Okrem požiadaviek, ktoré vyplývajú zo štruktúry a spôsobu, akým sú protokoly zapúzdrované, existujú aj funkčné požiadavky. Veľmi dôležitým parametrom sieťových zariadení je ich priepustnosť. Rýchlosť parsovania paketov musí odpovedať požiadavkám na priepustnosť, v opačnom prípade sa parser stane úzkym hrdlom celého zariadenia. Ďalšou dôležitou vlastnosťou je flexibilita, teda schopnosť pridať, odobrať alebo zmeniť protokol, ktorý je možné analyzovať. Flexibilita je potrebná hlavne z dôvodu vzniku nových a zmien v už existujúcich protokoloch. Bez možnosti prispôsobiť sa týmto zmenám, sa zariadenia rýchlo stávajú zastaralými a nepoužiteľnými.

3.2 Štruktúra protokolov

Pod pojmom štruktúra paketu, rozumieme poradie, v ktorom sú hlavičky protokolov zapúzdrené. Tieto informácie sú napr. v oblasti bezpečnosti využívané dohľadovými systémami, ktoré na základe štruktúry protokolov identifikuje potencionálny útok.

Pri tomto type analýz je potrebné pri procese parsovania paketov získavať tzv. *"štruktúrne metadáta"*. Príkladom štruktúrnych metadát je vzdialenosť dvoch polí, anglicky *offset*. Často sa využíva offset aplikačných dát, teda vzdialenosť konca hlavičky transportného protokolu od začiatku paketu. Ďalším typom štruktúrnych metadát je sekvencia identifikátorov hlavičiek, v poradí v akom boli rozpoznané, tzv. *"signatúra"*.



Obr. 3.1: Signatúra paketu zobrazená v rámci stromu protokolov

Signatúry, získané pri parsovaní, môžeme ukladať vo forme *stromu protokolov*. Jedná sa o abstraktný dátový typ, vychádzajúci z acyklického neorientovaného grafu. Uzly stromu reprezentujú hlavičky protokolov rozpoznané pri parsovaní paketu. Signatúra predstavuje cestu v strome smerom od koreňa k listovým uzlom. Na obrázku 3.1 je pod stromom protokolov schématicky znázornený paket. Signatúra tohoto paketu je v rámci stromu vyznačená červenými šípkami. Červená farba hlavičiek vo vnútri paketu, reprezentuje spôsob mapovania signatúry na uzly stromu. Ďalej si môžeme všimnúť, že strom protokolov môže obsahovať

viacero uzlov odpovedajúcich tomu istému protokolu, na obrázku 3.1 sú to IPv4, VLAN a Ethernet. Dôvodom je spôsob, akým je reprezentovaná štruktúra protokolov paketu a zároveň acyklickosť grafu, vyplývajúca z definície stromu. Tieto podmienky zaručujú, že strom protokolov, ktorý je konštruovaný postupne, podľa štruktúry paketov, nevytvára kombinácie signatúr, ktoré sa v prevádzke nevyskytli. Na obrázku sú šípkou s prerušovanou čiarou znázornené prepojenia, ktoré by značili takéto kombinácie.

3.3 Jazyk P4

Jazyk P4, (*"Programming Protocol-Independent Packet Processors"*), ako z názvu vyplýva, je jazyk určený na popis protokolovo nezávislých sieťových zariadení. Jedná sa o vysokoúrovňový deklaratívny jazyk popisujúci metódy a pravidlá práce s paketmi. Nedefinuje však spôsob vykonávania jednotlivých operácií [32]. Pôvodne bol jazyk P4 navrhnutý pre programovanie prepínačov, avšak s novými revíziami tohto jazyka, pribudli mnohé metódy a programovateľné bloky, ktoré umožňujú vytvárať popisy rôznych druhov sieťových zariadení.

Jazyk P4 je veľmi zaujímavý najmä kvôli jeho vlastnostiam, ako sú rekonfigurovateľnosť, vďaka ktorej je možné meniť spôsob spracovávania paketov aj po nasadení zariadenia. Ďalšou dôležitou vlastnosťou je protokolová nezávislosť, ktorá zabezpečuje, že zariadenie nie je viazané na špecifický formát protokolov, ale je možné ich popis kedykoľvek zmeniť. Poslednou vlastnosťou je nezávislosť na platforme, preto sa program v jazyku P4 nezaoberá špecifikami hardvérových častí. K tomuto účelu slúžia kompilátory, ktoré prevádzajú univerzálny popis, na závislý program určený pre konkrétnu platformu.

Jazyk P4 je určený k popisu dátovej roviny, zodpovedá teda za spracovávanie užívateľských dát prechádzajúcich zariadením. Spracovávanie paketov je popísané pomocou základných abstrakcií [34], ktoré jazyk P4 poskytuje:

- **Header types** popisujú formát hlavičiek protokolov ako postupnosť polí a ich dĺžok.
- **Parsers** špecifikuje povolené poradie hlavičiek v pakete a spôsob, akým je identifikovaná hlavička nasledujúceho protokolu.
- **Tables** asociuje užívateľom definované kľúče a akcie. P4 tabuľky umožňujú implementovať smerovacie tabuľky, ACL a tabuľky tokov.
- **Actions** sú fragmenty kódu definujúce akcie, ktoré môžu byť volané z rôznych častí programu.
- **Match-action** jednotky vykonávajú vyhľadávanie v tabuľkách a spúšťajú odpovedajúcu akciu.
- **Control flow** definujú poradie akcií a vyhľadávaní, ktoré budú vykonávané nad paketmi.
- **Metadata** sú štruktúry asociované s paketmi prechádzajúcimi cez dátovú rovinu. Môžu byť definované užívateľom ako aj výrobcom cieľovej platformy.
- **Extern objects** sú objekty špecifické pre každú platformu. Nie sú programovateľné, ale ich služby môžu byť volané prostredníctvom dobre špecifikovaného rozhrania.

Z pohľadu spracovávania sieťovej prevádzky, sú pre nás dôležité abstrakcie Header types a Parsers. Spolu, tieto časti jazyka P4, definujú funkcionalitu parseru paketov, ktorý je zodpovedný za rozpoznávanie hlavičiek protokolov prítomných v pakete a extrakciu potrebných položiek.

Popis štruktúry hlavičiek protokolov sa nachádza v časti *Header types*. Ako je možné vidieť na obrázku 3.2 definícia hlavičky protokolu začína kľúčovým slovom `header`. Potom nasleduje zoradená postupnosť položiek, ktoré hlavička obsahuje. Každá položka je definovaná veľkosťou a unikátnym názvom. Položky sú reprezentované ako polia, preto definícia veľkosti pozostáva z dátového typu, napr. bit a z počtu prvkov poľa. Táto hodnota je vo vnútri lomených zátvoriek. Jazyk P4 umožňuje v rámci protokolu označiť šírku poľa za variabilnú. V tom prípade je dátovým typom varbit a hodnota vo vnútri lomených zátvoriek určuje maximálnu veľkosť poľa.

```
header ethernet_t {
   bit<48> dstAddr;
   bit<48> srcAddr;
   bit<16> etherType;
}
```

```
header IPv4_h {
   bit<4> version;
   ...
   bit<32> srcAddr;
   bit<32> dstAddr;
   varbit<320> options;
}
```

Obr. 3.2: Definície hlavičiek protokolov Ethernet a UDP, v jazyku $P4_{16}$

Na obrázku 3.3 je znázornený popis jednoduchého parsera. Definícia samotného parsera začína hlavičkou, ktorá sa skladá z kľúčového slova `parser`, názvu a zoznamu vstupných a výstupných parametrov. Nasleduje telo parsera, ktoré obsahuje definície jeho stavov. Každá definícia stavu začína kľúčovým slovom `state`, nasleduje unikátny názov a telo stavu. Funkcionalitu implementovanú v tele stavu môžeme rozdeliť na dve fázy.

```
parser ParserImpl(packet_in packet, out headers hdr,
                  inout metadata meta, inout metadata_t metadata)
{
       state start {
              transition parse_ethernet;
       }
       state parse_ethernet {
              packet.extract(hdr.ethernet);
              transition select(hdr.ethernet.etherType) {
                     0x800: parse_ipv4;
                     default: accept;
              }
       }
}
```

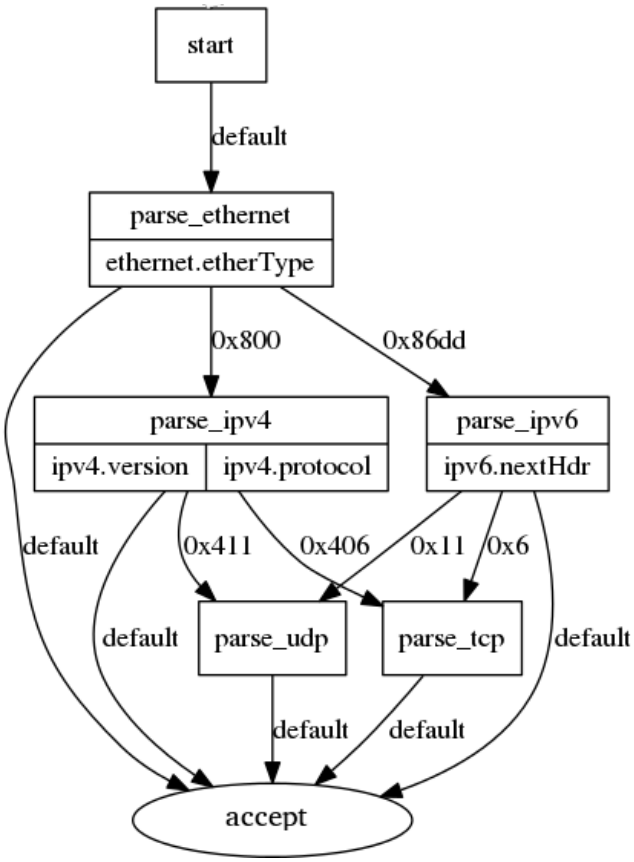
Obr. 3.3: Časť definície parsera v jazyku $P4_{16}$

Prvou fázou je extrakcia položiek, ktorá je reprezentovaná volaním metódy `extract()` nad paketom na vstupe. Táto fáza môže byť vynechaná, ako na obrázku 3.3 v stave start.

Druhou fázou je prechod do ďalšieho stavu. Za kľúčovým slovom `transition`, nasleduje názov nasledujúceho stavu alebo rozhodovací blok `select`, ktorý sa použije v prípade, že výber nasledujúceho stavu je ovplyvnený hodnotou niektorého poľa hlavičky, tak ako to je v stave `parse_ethernet`. Hodnota poľa, ktorého identifikátor obsahuje volanie `select`, bude porovnávaná s konštantami uvedenými pred dvojbodkami vo vnútri rozhodovacieho bloku. Prvá zhoda pri porovnaní určí stav, ktorý prevezme riadenie. Jeho názov sa nachádza za dvojbodkou. Špeciálna hodnota `default` určuje východzí stav, do ktorého parser prejde, ak nedôjde ku zhode ani s jednou z prechádzajúcich hodnôt.

Proces parsovania paketu, ktorý odpovedá postupnosti zmien stavov a podmienok, pri ktorých sa tieto zmeny vykonávajú, môžeme znázorniť ako tzv. *"graf protokolov"*, angl. *parse graph* [34]. Každý stav parseru je reprezentovaný jedným uzlom grafu protokolov.

Obrázok 3.4 obsahuje parse graph odpovedajúci popisu jednoduchého parseru, ktorého počiatočným stavom je `start` a parsovanie paketu je ukončené v stave `accept`. Uzly grafu sú zobrazené ako obdĺžniky, ktoré obsahujú názov stavu. Ak je zmena stavu podmienená hodnotou poľa v hlavičke protokolu, jeho názov je uvedený pod čiarou v spodnej časti obdĺžnika, ako napr. v prípade uzlu `parse_ethernet`. Hodnota uvedená pri šípke reprezentujúcej prechod, určuje hodnotu poľa zdrojového uzlu, pri ktorej sa vykoná prechod do cieľového uzlu.



Obr. 3.4: Graf protokolov reprezentujúci konečný automat popísaný v jazyku P4

Ďalej si môžeme všimnúť, že strom protokolov odpovedá jednotlivým postupnostiam uzlov parse grafu, ktorými prejde parser paketov. Tým pádom je možné pri parsovaní paketov používať jazyk P4 a z toho plynúce výhody, akými sú flexibilita a protokolová nezávislosť

a zároveň spolu s ostatnými metadátami získavať aj informácie o štruktúre spracovaných paketov.

Vzhľadom na potrebu dosiahnuť vysokú priepustnosť, je vhodné pri implementácii jednotky, vykonávajúcej analýzy hlavičiek protokolov a extrakcie polí, využiť hardvérovú akceleráciu. Túto voľbu podporuje aj fakt, že každý graf protokolov môžeme previesť na ekvivalentný **deterministický konečný automat**. Preto je možné na základe parseru popísaného v jazyku P4, vytvoriť konečný automat popísaný v jazyku VHDL a ten mapovať do technológie FPGA. Je však potrebné navrhnúť architektúru analyzačnej jednotky tak, aby výsledná implementácia bola funkčne ekvivalentná s popisom v jazyku P4.

Kapitola 4

Mapovanie jazyka P4 do VHDL

Pre dosiahnutie priepustnosti 10 Gb a flexibility, ktorá je pri súčasnom trende zmien a vzniku nových sieťových protokolov nevyhnutnosťou, je výhodné využiť potenciál jazyka P4 v kombinácií s technológiou FPGA. Aby sme mohli využiť hardvérovú akceleráciu a možnosť rekonfigurácie hradiel FPGA čipu, spolu s flexibilitou, ktorú prináša popisovanie spracovania paketov v jazyku P4, potrebujeme vhodný mapovací nástroj medzi jazykmi P4 a VHDL.

V súčasnej dobe existuje niekoľko komerčných nástrojov, napr. od firmy Netcope [24] alebo združenia CESNET [2]. Oba tieto nástroje však slúžia k popisu prechodu paketu sieťovým zariadením, kde analýza hlavičiek je iba začiatkom celého procesu. Výstupom týchto nástrojov sú zreťazené komponenty komunikujúce prostredníctvom špecifických rozhraní. Navyše samotný jazyk P4 neobsahuje podporu exportu štruktúry paketov.

Z dôvodu nedostupnosti vhodných nástrojov vznikol v rámci mojej bakalárskej práce [32] generátor, ktorý mapuje jazyk P4 do jazyka VHDL. Rozhodol som sa ho použiť v tejto práci, pretože na rozdiel od ostatných nástrojov je určený ku generovaniu jednotiek spracovávajúcich hlavičky protokolov. Tým pádom je možné jeho rozhrania prispôsobiť návrhu celého systému. Táto kapitola okrem opisu architektúry jednotky HFE, skratka z angl. *header field extraction* a spôsobu mapovania jazyka P4 do VHDL, obsahuje popis potrebných úprav generátoru.

4.1 Hardvérová architektúra

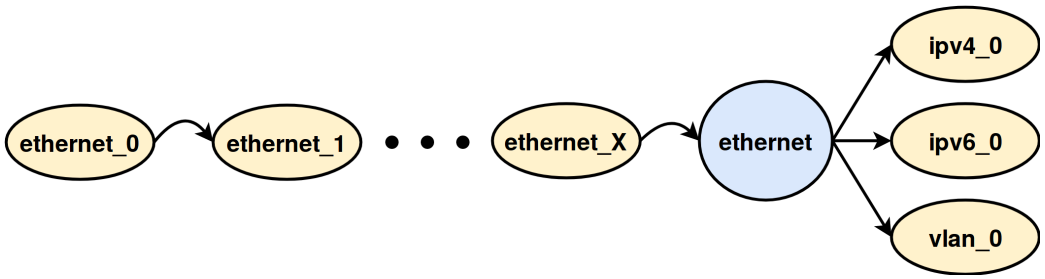
Navrhovaná jednotka identifikuje polohu polí hlavičiek protokolov v rámci paketu, ich hodnotu dočasne uloží a na konci spracovania odošle. Vstupom jednotky je nepretržitý tok dát vo forme paketov, ktoré sú spracovávané po jednom. Výstupom parsovania každého paketu jednotkou HFE je štruktúra extrahovaných metadát.

Jadrom celej jednotky je modul FSM. Analyzuje hlavičky protokolov a extrahuje z nich položky. Tento modul je celý generovaný na základe popisu protokolov a ich vzájomných prepojení v jazyku P4. Modul FSM obsahuje konečný automat, ktorý riadi proces parsovania paketu. Parsovanie hlavičky protokolu nemusí byť realizované jedným, ale celou postupnosťou stavov. Počet stavov v postupnosti závisí na štruktúre hlavičky a počte polí, potrebných pri spracovaní hlavičky. Analýzu hlavičky protokolu je možné rozdeliť do dvoch fáz [32]:

- **Fáza načítania dát** začína prvým a končí predposledným stavom postupnosti. Poradie stavov v tejto fáze je napevno stanované generátorom. V každom stave sa načítavajú dáta, ktoré sú spracované v nasledujúcom stave. Ak načítané dáta obsahujú

informácie potrebné pri identifikácii nasledujúcej hlavičky, sú dočasne uložené v pomocných registroch. Na obrázku 4.1 sú stavy, ktoré patria do fázy načítania dát, vyobrazené ako žlté ovály.

- **Fáza vetvenia** prebehne vždy v poslednom stave postupnosti. Na základe porovnania dát uložených v pomocných registroch s hodnotami definovanými v zdrojových súboroch jazyka P4 sa rozhodne, ktorá hlavička protokolu v pakete nasleduje. Ak sme práve ukončili spracovanie poslednej hlavičky v pakete, bude vykonaný prechod do koncového stavu automatu. Na obrázku 4.1 je fáza vetvenia zobrazená modrým kruhom.



Obr. 4.1: Postupnosť stavov vykonávajúca parsovanie hlavičky protokolu Ethernet [32]

Rovnako ako graf protokolov aj konečný automat, ktorý je súčasťou jednotky HFE, obsahuje stavy, ktoré sa nezúčastňujú na spracovaní hlavičiek. Tieto stavy riadia rutiny, ktoré sa vykonávajú vždy na začiatku a na konci analýzy paketu [32].

- Počiatočný stav automatu, `start`, riadi inicializačnú rutinu spracovania paketu. Keďže parsovanie je nezávislé na predchádzajúcich paketoch, táto rutina inicializuje všetky pomocné registre a pamäte určené k uchovávaniu metadát. V prípade, že na vstupe nie sú žiadne pakety, konečný automat zotrváva v tomto stave.

- Stav `fin` implementuje vykonávanie rutiny, po úspešnom ukončení parsovania poslednej hlavičky v pakete. Odpovedá stavu `accept` z P4 parseru. Táto rutina exportuje metadáta zhromažďované počas analýzy paketu. Potom konečný automat prechádza opäť do stavu `start`.

- Druhou ukončovacou rutinou je stav `error`, ktorý sa vykoná v prípade výskytu chyby. Chyba je vyvolaná pri načítavaní dát paketu v prípade, že požadovaná adresa neobsahuje platné dáta. Spracovávanie paketu je ihneď ukončené, exportujú sa aktuálne metadáta a taktiež sa prejde do stavu `start`, kde sa začína so spracovaním ďalšieho paketu.

4.2 Softvérový generátor

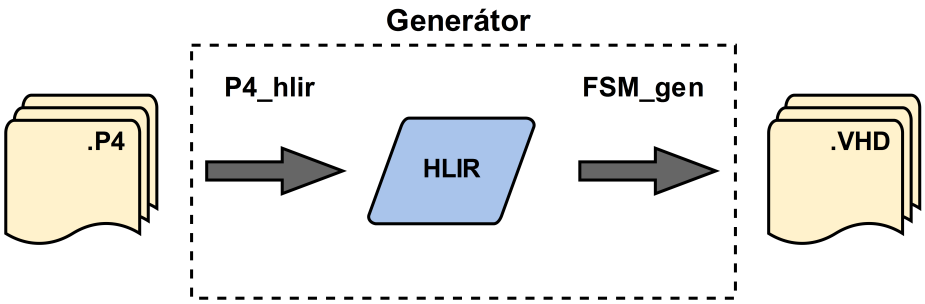
Flexibilnosť, ktorú vyžadujeme prináša rekonfigurovateľnosť hradiel technológie FPGA. Aby bolo možné, zmeny v zdrojových súboroch jazyka P4 dostatočne rýchlo premietnuť do konfigurácie jednotky, je nutné použiť generátor do jazyka VHDL.

Proces prekladu je implementovaný generátorom, ktorý sa skladá z dvoch podprogramov: `P4_hlir` a `FSM_gen`. Na obrázku 4.2 sú reprezentované šípkami, ktorých orientácia rozlišuje ich vstupy a výstupy. Súčasťou generátoru sú tiež objekty internej reprezentácie,

`HLIR`, ktoré sú dočasne uložené v operačnej pamäti. Tieto objekty navzájom prepájajú oba podprogramy.

Program `P4_hlir` [33] bol vyvinutý ako otvorený softvér, ktorý je súčasťou projektu *P4 language*, ktorý riadi organizácia *P4 Language Consortium*. Tento program vykonáva lexikálnu a syntaktickú analýzu súborov v jazyku P4. Analýzy určujú, či vstupné súbory odpovedajú špecifikácií jazyka P4 z lexikálneho a syntaktického hľadiska.

Druhou časťou generátora je podprogram `FSM_gen`, ktorý vykonáva sémantickú analýzu a generovanie kódu. Jeho vstupmi sú objekty HLIR, nad ktorými prebiehajú sémantické analýzy, ktoré zaisťujú, že všetky využité konštrukcie jazyka P4 je možné reprezentovať v navrhnutej architektúre. Na záver prebehne generovanie kódu, z čoho vyplýva, že výstupom programu FSM_gen sú súbory v jazyku VHDL [32].



Obr. 4.2: Postup generovania popisu v jazyku VHDL z jazyka P4 [32]

Stavy parsera v jazyku P4 spracovávajú jednotlivé hlavičky protokolov. Preto je v prvom kroku analyzovaná ich funkcionalita a výsledkom sú im prislúchajúce objekty. Tieto objekty obsahujú informácie o krokoch vykonávaných pri spracovávaní konkrétneho protokolu.

V ďalšom kroku generovania sú stavy parsera prevedené na graf protokolov. Uzly tohto orientovaného grafu reprezentujú parsovanie založené na inštanciách hlavičiek. V prípade, že hlavička jedného typu vytvára niekoľko inštancií, tak aj parse graf obsahuje viacero zreťazených stavov. Každý z týchto stavov vykonáva analýzy nad jednou inštanciou hlavičky. Týmto spôsobom môžeme zhora ohraničiť počet hlavičiek rovnakého protokolu, ktoré môžu vo validnom pakete nasledovať za sebou.

V poslednom kroku je parse graf mapovaný do architektúry HFE. Abstrakcia uzlu parse grafu analyzujúceho inštanciu hlavičky protokolu, je nahradená postupnosťou stavov, ktoré spolu tvoria presný postup parsovania tejto hlavičky. Zároveň v tomto kroku prebieha generovanie kódu, počas ktorého je popis činností v jednotlivých stavoch konečného automatu vyjadrený v jazyku VHDL [32].

4.3 Dosiahnutie priepustnosti 10 Gbps

Jednotka HFE, ktorá bola popísaná v predchádzajúcich dvoch sekciách je schopná spracovávať pakety na linkách s priepustnosťou 1 Gb. Za účelom zvýšenia maximálnej priepustnosti na 10 Gb a minimalizácie réžie spracovania hlavičky protokolu, bolo nutné vykonať niekoľko dôležitých úprav v generátore. Aby ani v extrémnych prípadoch, ktoré z hľadiska parsovania paketov znamenajú veľký počet hlavičiek v paketoch malej veľkosti, nedochádzalo ku znižovaniu priepustnosti.

- **Bitová šírka** v súčine s hodinovou frekvenciou určuje maximálnu teoretickú priepustnosť jednotky. Pre každé použitie je nutné nájsť vhodný pomer týchto dvoch veličín, pričom musíme zohľadniť aj parametre konkrétneho FPGA čipu. Preto prvou z nutných úprav bola užívateľom nastaviteľná bitová šírka vstupného a výstupného rozhrania.

- **Špekulatívne načítavanie dát** znižuje počet stavov konečného automatu v sekvencii spracovávajúcej hlavičku protokolu. Prvý stav v sekvencii slúžil iba na načítavanie dát. Ak posledný stav každej sekvencie špekulatívne načíta dáta zo začiatku ďalšej hlavičky a v nasledujúcej sekvencii stavov tieto dáta rovno spracujeme, tak znížime réžiu parsovania každej hlavičky. Toto opatrenie zamedzuje poklesu priepustnosti v extrémnych prípadoch.

- **Optimalizácia načítavania dát** zabraňuje opakovanému načítavaniu tých istých dát. Dodatočné heuristiky menia poradie načítavaných dát tak, aby pri ich spracovaní boli naraz vykonané všetky potrebné operácie, napr. výpočet premenlivej dĺžky a dočasné uloženie dát kvôli exportu. Okrem toho, je preusporiadanie vykonávané s ohľadom na zvyšovanie efektivity špekulatívneho načítavania dát.

- **Štruktúra paketu** je daná postupnosťou hlavičiek protokolov. V pôvodnej verzii nebolo možné tieto informácie z HFE exportovať. Po tejto úprave, jednotka HFE exportuje postupnosť 20 identifikátorov protokolov, v poradí v akom boli jednotlivé hlavičky rozpoznané.

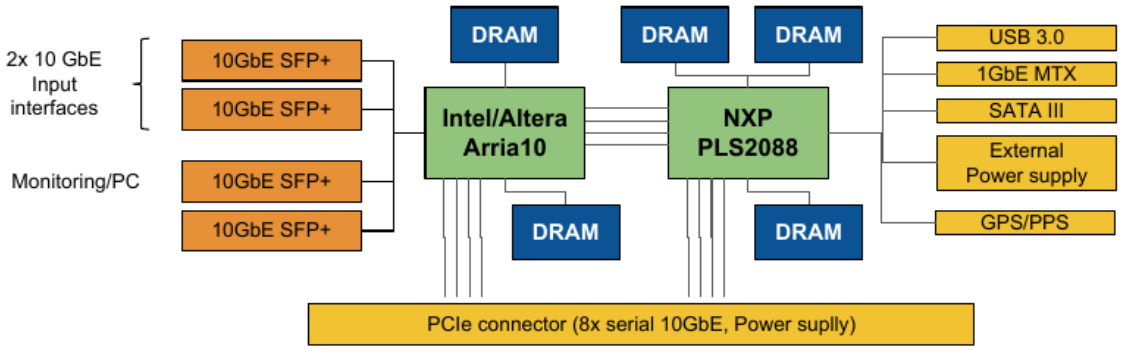
Po úpravách, ktoré som navrhol a implementoval, generátor z jazyka P4 do VHDL, umožňuje generovanie jednotky HFE, ktorá spĺňa stanovené kritéria priepustnosti a potrebných exportovaných dát. Dôsledkom toho poskytuje všetky informácie, ktoré potrebujeme pri monitorovaní sieťových protokolov. Zároveň vďaka hardvérovej akcelerácií v FPGA umožňuje dosiahnuť nielen priepustnosť 10 Gb, ale aj znižuje záťaž procesoru a tak šetrí výpočtový výkon pre ďalšie analýzy.

Kapitola 5

Hardvérová platforma

Výber vhodnej platformy je pri návrhu zariadenia veľmi dôležitý, pretože fyzické komponenty a spôsob ich prepojenia limitujú výkon a vlastnosti výsledného systému. Rovnako je dôležité výber platformy podmieniť požiadavkami, ktoré vyplývajú z prípadov použitia konkrétneho systému.

Za účelom monitorovania sieťových protokolov vznikla nová platforma. Jej schéma je znázornená na obrázku 5.1 a skladá sa zo štyroch typov blokov.

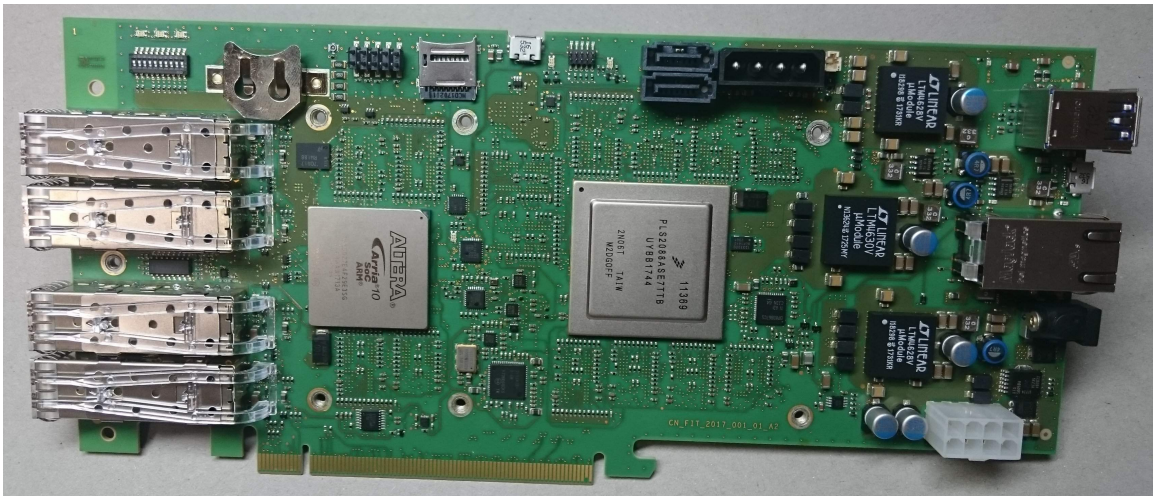


Obr. 5.1: Schematický popis platformy monitorovacieho systému

- **Výpočtový blok** je reprezentovaný zelenými obdĺžnikmi. Prvým výpočtovým blokom je FPGA čip od firmy Intel z rodiny Arria 10 GX 270 [37]. Ide o stredne veľký čip, ktorý obsahuje 12 transcieverov [16] a umožňuje dosiahnuť frekvenciu až 500 MHz [17]. Druhým výpočtovým blokom je výkonný procesor z rodiny QorIQ LS-2 od spoločnosti NXP [38]. Tento procesor, obsahujúci osem 64-bitových ARM Cortex-A72 jadier. Bol špeciálne navrhnutý pre sieťové aplikácie, preto obsahuje množstvo akceleračných jednotiek napr. pre pattern-matching a kompresiu dát.
- **Bloky DRAM** sú pripojené ku obom výpočtovým blokom. Na obrázku 5.1 sú znázornené modrými obdĺžnikmi a slúžia ako dynamické pamäte.
- **Vstupno/výstupné rozhrania** sú vyobrazené ako oranžové obdĺžniky. Jedná sa o transcievery typu SFP+. Dva z nich sú vstupnými rozhraniami, ktorými prichádzajú pakety zo siete. Ďalšie dva sú výstupné rozhrania, vyhradené pre export paketov, ktoré sú určené k uloženiu alebo ďalším analýzam mimo monitorovacie zariadenie.

- **Rozhrania periférnych zariadení** rozširujú prípady použitia platformy a sú reprezentované žltými obdĺžnikmi. Využívajú sa najmä pri manažmente systému, ako napr. 1 Gb ethernetové rozhranie, alebo pri ukladaní dát na disk, rozhranie SATA3. PCI express konektor, znázornený pri spodnom okraji obrázku, nie je určený ku komunikácii medzi výpočtovými blokmi, ale k napájaniu celého zariadenia.

Z jednoduchého nákresu zobrazujúceho najdôležitejšie bloky zariadenia, vznikla postupnou špecifikáciu komponent podrobná schéma. Podľa nej bola vyrobená doska tlačených spojov, angl. *printed circuit board*, na ktorú boli osadené externé súčiastky. Na obrázku 5.2 je odfotená hotová platforma po osadení procesoru, FPGA čipu a ostatných komponent.



Obr. 5.2: Fotografia hotovej platformy

Kapitola 6

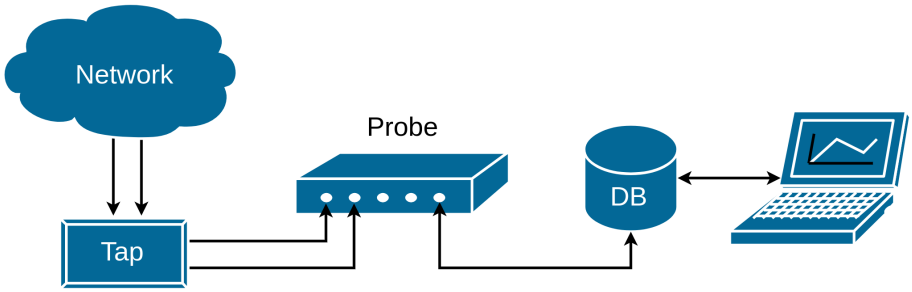
Návrh monitorovacieho systému

V úvode kapitoly zhrniem požiadavky kladené na monitorovací systém. Ďalej bude nasledovať popis jeho jednotlivých častí a spôsobu ich prepojenia. V podkapitolách sa budem venovať identifikácii a rozdeleniu funkčných požiadaviek medzi konkrétne časti monitorovacieho systému. Na koniec sa prepracujem až k popisu mapovania jednotlivých úloh na výpočtové prostriedky a spôsob ich implementácie.

Funkčné požiadavky, ktoré má navrhovaný monitorovací systém spĺňať sú priamo formulované v zadaní tejto diplomovej práce. Ďalšie požiadavky z nich nepriamo v ychádzajú a boli identifikované pomocou informácií, ktoré sú popísané v predchádzajúcich kapitolách. Na základe analýzy problematiky som definoval nasledujúce požiadavky:

- **Flexibilita systému** je definovaná typom zmien, ktorým je možné systém prispôsobiť. V oblasti počítačových sietí najčastejšie dochádza k vzniku nových a zmenám v už existujúcich sieťových protokoloch. Bez možnosti prispôsobenia, by sa mohol celý monitorovací systém stať nepoužiteľným.
- **Monitorovanie tokov** rozdeľuje a agreguje sieťovú prevádzku do skupín, ktoré sa líšia v hodnotách skúmaných položiek. Tým pádom poskytuje kvalitatívne orientovaný pohľad na dianie v sieti. Monitorovanie tokov vykonávame za účelom získavania základných informácií o prebiehajúcej komunikácií, tak aby sme tieto informácie mohli v prípade potreby využiť pri diagnostike, detekcii nežiadúcej prevádzky alebo účtovaní. Dôležité je, aby informácie, ktoré o tokoch získavame a uchovávame, odpovedali informáciám, ktoré potrebujeme vzhľadom na splnenie cieľov monitoringu.
- **Strom protokolov** znázorňuje štruktúru zapúzdrenia protokolov v paketoch. Uzly stromu reprezentujú hlavičky protokolov rozpoznané pri parsovaní paketu. Pozícia toku v rámci stromu protokolov rozširuje samotný pojem sieťový tok o štruktúrne informácie a tak umožňuje dosiahnuť jemnejšiu granularitu monitorovania sieťovej prevádzky. Taktiež štatistické údaje môžu byť získavané pre každý uzol stromu, čo je možné využiť napr. pri správe sietí a plánovaní zmien v ich architektúre.
- **Filtrovanie paketov** sa využíva pri záchyte časti prevádzky, o ktorú máme záujem. Filtrovaním postupne prechádzajú všetky pakety, pričom dochádza k porovnávaniu hľadaných identifikátorov s hodnotami z hlavičiek protokolov. Každý z identifikátorov, je buď porovnávaný na zhodu, alebo je porovnávanie preskočené na základe nastavenej masky. Pakety, ktoré odpovedajú zadanému filtrovaciemu pravidlu môžme uložiť za účelom dôkladnejších a časovo náročnejších analýz, ktoré je možné vykonávať len nad súborom dát obmedzenej veľkosti.

- **Perzistentné ukladanie dát** musí podporovať každý monitorovací systém, ktorého výstupy môžu byť v budúcnosti potrebné. Tým pádom môže užívateľovi poskytnúť historické dáta, ktoré sú nenahraditeľné pre dosiahnutie cieľov monitorovania. Na základe historických dát je možné napr. identifikovať zdroj podozrivej prevádzky, ktorá je prítomná na sieti od určitej udalosti. V našom prípade monitorovací systém vytvára 3 typy výstupov: záznamy o tokoch, stromy protokolov a odfiltrované pakety.
- **Užívateľské rozhranie** slúži nielen pri prezentácií výstupov monitorovacieho systému, ale môže aj sprostredkovať interakciu užívateľa so systémom. Zároveň možnosť konfigurácie interných komponent a teda zmeny funkcionality, vhodne dopĺňa požiadavku flexibility.



Obr. 6.1: Schéma zapojenia monitorovacieho systému

Keďže monitorovací systém je založený na monitorovaní tokov, bude spôsob jeho nasadenia podobný schémam protokolov NetFlow a IPFIX. Na obrázku 6.1 môžeme vidieť, že monitorovacie zariadenie, nazývané **sonda**, je do siete, na obrázku znázornenej oblakom, pripojená prostredníctvom zariadenia `Tap`. Toto zariadenie preposiela celú sieťovú prevádzku do sondy, na obrázku označenej `Probe`. Sonda vykonáva samotnú monitorovaciu činnosť a získané informácie odosiela na úložisko dát, ktoré je na obrázku reprezentované symbolom databázy s označením `DB`. K uloženým dátam je možné pristupovať z počítača, ktorý slúži k ich vizualizácií prostredníctvom užívateľského rozhrania.

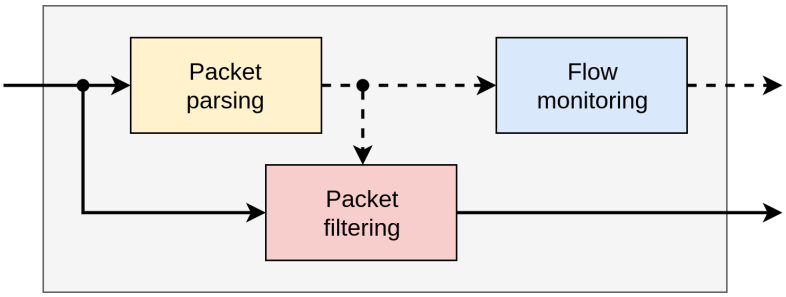
Monitorovací systém sa skladá z dvoch výpočtových častí, medzi ktoré sú rozdelené jednotlivé funkčné požiadavky. V nasledujúcom texte budú tieto časti popísané z hľadiska funkcionality, návrhu a implementácie.

6.1 Sieťová sonda

Monitorovacie zariadenie, ktoré je určené pre nasadenie v 10 Gb sieťach, musí spĺňať nielen požiadavky na priepustnosť, ale aj na veľkosť generovaných dát. Zvoliť vhodný stupeň agregácie je dôležité, nie len z dôvodu výpočtovej náročnosti analýz veľkého objemu uložených dát, ale aj pre problémy s ich priebežným ukladaním. Pri návrhu je tiež dôležité myslieť na veľkosť komunikácie jednotlivých častí systému. Preto je potrebné monitorovací systém rozdeliť na časť, ktorá spracováva sieťovú prevádzku a úložisko, tzv. *“mediačné zariadenie“* alebo *“kolektor“*. Tým pádom budú všetky komponenty, ktoré pracujú s paketmi alebo ich metadátami, združené v rámci jednej časti monitorovacieho systému.

Tomuto popisu odpovedá práve sonda, implementujúca monitorovanie tokov a filtrovanie paketov. Na obrázku 6.2 môžeme vidieť schému zapojenia základných funkčných blokov sondy. Okrem už spomínaného monitorovania tokov, na obrázku označeného modrým

a filtrovania paketov, označeného červeným obdĺžnikom sme pridali žltý blok vykonávajúci analýzy a extrakcie paketov. Táto komponenta získava z paketov metadáta, ktoré sú potrebné pre prácu zvyšných dvoch komponent. Komunikácia komponent je na obrázku 6.2 znázornená pomocou šípok. Druh prenášaných dát je rozlíšený podľa typu čiary. Plná čiara reprezentuje prenos celých paketov a prerušovaná metadáta a záznamy o tokoch.



Obr. 6.2: Konceptuálna schéma fungovania sondy

Keďže sme už identifikovali úlohy, ktoré budú na sonde realizované a poznáme aj hardvérovú platformu sondy, môže prejsť k rozdeleniu a mapovaniu jednotlivých úloh na výpočtové zdroje dostupné na platforme. Na základe informácií z kapitoly 5 vieme, že platforma sondy obsahuje dve výpočtové jednotky:

- **FPGA čip** je určený na akceleráciu časovo kritických operácií. V oblasti sietí sú nimi najmä operácie vykonávané s príchodom každého paketu. Preto je vhodné v FPGA implementovať úlohy, ktoré pracujú so vstupnými paketmi. Toto rozhodnutie podporuje aj fakt, že máme k dispozícii nástroj schopný generovať jednotku HFE s priepustnosťou 10 Gbps. V technológii FPGA je vhodné implementovať parsovanie a filtrovanie paketov. Tieto dve úlohy sú úzko previazané aj preto, že pri filtrovaní paketov sa porovnávajú metadáta získané pri parsovaní s užívateľom zadanými pravidlami.
- **Procesor** je vhodný na vykonávanie komplexnejších a pamäťovo náročných úloh, medzi ktoré patrí monitorovanie tokov. Aj keď objem extrahovaných metadát môže v extrémnych prípadoch presiahnuť veľkosť paketov, z ktorých boli metadáta získané, tak sústavná práca s rôznymi dátovými štruktúrami favorizuje procesor na vykonávanie tejto úlohy.

6.1.1 Hardvérová architektúra

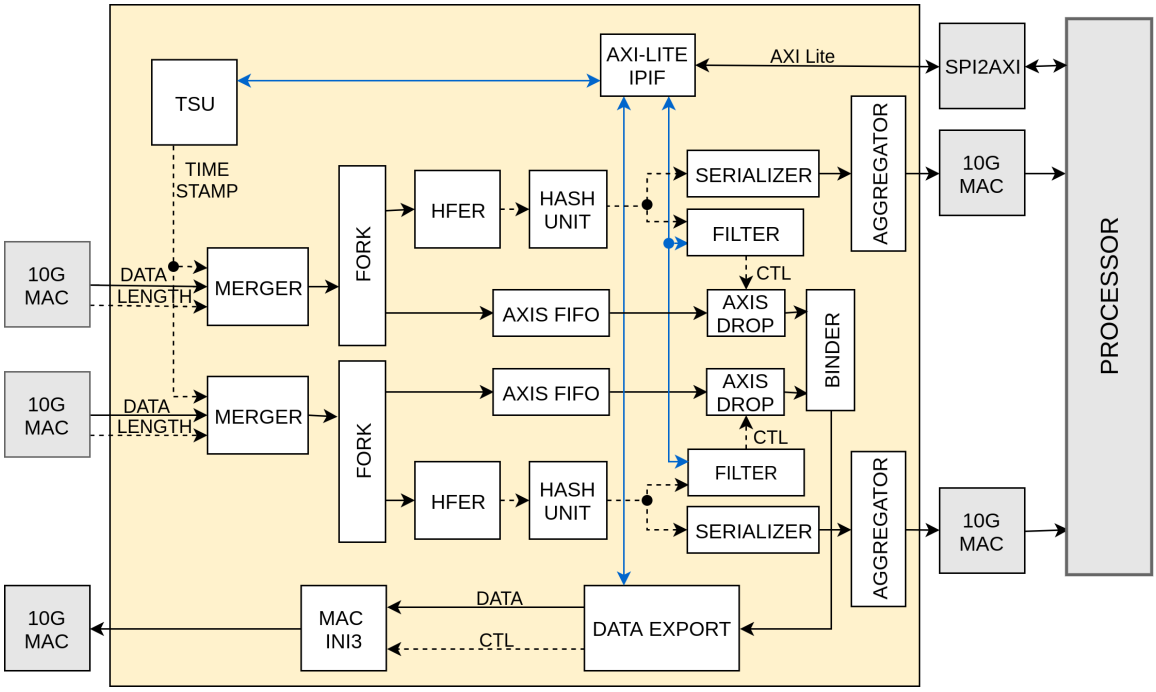
Časť sieťovej sondy, ktorá v FPGA obsahuje implementácie hardvérovo akcelerovaných výpočtov, bude v tomto texte označovaná ako tzv. *"hardvérová časť"*. Jej súčasťou musia byť okrem komponent implementujúcich parsovanie a filtrovanie paketov aj ďalšie komponenty, ktoré zodpovedajú za príjem sieťovej prevádzky, odosielanie výstupných dát a konfiguráciu komponent.

Na obrázku 6.3 sú v žltom poli znázorné komponenty hardvérovej časti sondy a ich vzájomné prepojenie. Ku fyzickým rozhraniam sú pripojené komponenty `10G MAC`, ktoré implementujú časť linkovej vrstvy. Platforma umožňuje využiť dve vstupné rozhrania. Aby bolo možné pri spracovaní dát dosiahnuť linkovú rýchlosť, každému vstupnému rozhraniu priradíme vlastnú zreťazenú linku, nazývanú angl. *pipeline*. Jej prvou komponentou je `Merger`, ktorý paketu z dátového vstupu predradí špeciálnu hlavičku, ktorá obsahuje časovú známku,

z komponenty `TSU`, identifikátor vstupného rozhrania a veľkosť paketu. Takto rozšírený paket je odoslaný komponente `Fork`, ktorá paket zduplikuje a tak vytvára dve vetvy v rámci jednej zreťazenej linky.

Prvá vetva je analytická, pretože obsahuje komponentu `HFE`, ktorá parsuje pakety a získava z nich potrebné metadáta. Extrahované **metadáta** delíme do dvoch skupín: **základné** a **štatistické**. Medzi základné metadáta patrí zdrojová a cieľová IP adresa, zdrojový a cieľový port, L4 protokol, verzia IP protokolu, dva VLAN tagy, signatúra a identifikátor vstupného rozhrania. Medzi štatistické metadáta patria veľkosť paketu a časová značka jeho prijatia. K nim je komponentou `Hash unit` pripojený aj odtlačok, angl. *hash*, vygenerovaný na základe základných metadát. Nasleduje komponenta `Serializer`, ktorá mení spôsob prenosu metadát z paralelného na sériový. Niekoľko serializovaných transakcií je, z dôvodu zníženia réžie prenosu na procesor, spojených komponentou `Aggregator` a následne odoslaných cez 10 Gbps rozhranie. Komponentu `Filter` je možné konfigurovať zo softvéru. Výstupom filtra je riadiaci signál, ktorý signalizuje, či paket odpovedá nakonfigurovanému pravidlu.

Druhá vetva zreťazenej linky slúži na export celých paketov zo zariadenia. Pakety sú dočasne uložené v komponente `AXIS FIFO`, ktorej výstup je riadený komponentou `AXIS DROP`, zabezpečujúcou zahadzovanie paketov, ktorá je inštruovaná už spomínaným filtrom. V tomto bode sú obe zreťazené linky spojené spoločnou komponentou `Binder`, ktorá pakety určené k exportu spája do jedného toku.



Obr. 6.3: Podrobná schéma prepojenia hardvérových komponent v hardvérovej časti sondy

Komponenty v rámci hardvérovej časti navzájom komunikujú prostredníctvom niekoľkých typov rozhraní, ktoré sú na obrázku 6.3 graficky rozlíšené. Plnou čiernou čiarou sú značené sériové rozhrania typu **AXI4Stream**. Ďalším typom použitých rozhraní sú paralelné synchrónne rozhrania, na obrázku znázornené prerušovanou čiarou. Prostredníctvom týchto rozhraní sú prenášané: metadáta medzi jednotkami HFER, Hash unit, Serializer a Filter, kontrolné signály označené `CTL`, časové značky z jednotky TSU nazvané `Time Stamp`

a veľkosti paketov odosielané z komponent MAC 10G označené ako `Length`. Posledným je rozhranie IPIF, ktoré slúži na konfiguráciu komponent. Na obrázku je znázornené plnou modrou čiarou a využíva sa napr. pri nastavovaní aktuálneho času jednotky TSU alebo užívateľom vybratých filtrovacích pravidiel do komponenty Filter.

S cieľom dosiahnuť priepustnosť na úrovni linkovej rýchlosti, musíme vhodne zvoliť šírku dátových ciest a pracovnú frekvenciu obvodu. Maximálna bitová rýchlosť 10 Gbps linky je dosiahnutá pri prenose paketov s maximálnou veľkosťou, ktorá je 1500 bajtov. Za týchto podmienok je pomer réžie vzhľadom k veľkosti dát najmenší. Veľkosť réžie paketu, ktorú tvorí preambula, kontrolný súčet a medzipaketová medzera je 24 bajtov. Maximálnu potrebnú priepustnosť systému, ktorú musíme dosiahnuť pre každý vstupný port vypočítame ako pomer veľkosti dátovej časti a veľkosti celého rámcu s réžiou, vynásobený linkovou rýchlosťou.

$$1 * 10^{10}\ bit/s\ *\ \frac{1500\ byte}{(1500\ byte\ +\ 24\ byte)}\ =\ 9,85 * 10^{9}\ bit/s \tag{6.1}$$

Na základe toho sme stanovili šírku dátových zberníc na 64 bitov a pracovnú frekvenciu obvodu na 156 MHz. S týmito parametrami je priepustnosť jednej dátovej cesty v systéme rovná:

$$64\ bit\ *\ 1,56 * 10^{8}\ Hz\ =\ 9,98 * 10^{9}\ bit/s \tag{6.2}$$

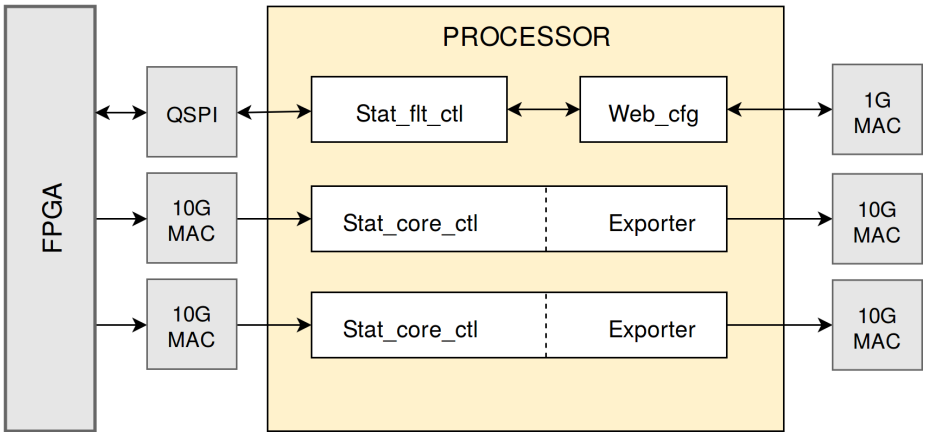
Čo je viac ako maximálna potrebná priepustnosť, ktorú sme vypočítali a preto je priepustnosť dátovej cesty hardvérovej časti dostatočná.

Ako bolo na začiatku tejto kapitoly spomenuté, jednotlivé komponenty hardvérovej časti tvoria dve ekvivalentné zreťazené linky. Každá komponenta tvoriaca jeden stupeň zreťazenej linky obsahuje pole výstupných registrov. Keďže čas potrebný na spracovanie paketu jednotlivými stupňami zreťazenej linky je maximálne rovný času priechodu paketu dátovou cestou, priepustnosť zreťazenej linky je dostatočná.

6.1.2 Softvérová architektúra

Pojmom *"softvérová časť"* bude v tomto texte označovaná skupina úloh vykonávaných na procesore sondy. Všetky tieto úlohy vychádzajú zo štyroch základných požiadaviek popísaných na začiatku tejto kapitoly. Jedná sa o monitorovanie tokov, filtrovanie paketov, flexibilitu systému a perzistentné ukladanie dát. Aj keď žiadna z funkčných požiadaviek okrem monitorovania tokov nebola pôvodne mapovaná do softvérovej časti sondy, je nutné sprostredkovať komunikáciu medzi hardvérovou a softvérovou časťou a medzi sondou a kolektorom.

Na obrázku 6.4 môžeme vidieť blokovú schému softvérovej časti sondy. Zo vstupných rozhraní sú metadáta prevzaté procesom `Stat_core_ctl`, ktorý ich agreguje vo forme záznamov o tokoch. Tie sú neskôr odoslané časťou tohoto procesu, označenou `Exporter`, cez výstupné rozhrania sondy, ktoré sú umiestnené pri pravom okraji obrázku. Môžeme si všimnúť, že tak ako v hardvérovej časti má každé vstupné rozhranie dedikovanú zreťazenú linku, rovnako aj v softvérovej časti sú spustené dve inštancie procesu Stat_core_ctl, každá vyhradená pre jedno rozhranie. Interakciu medzi užívateľom a systémom sprostredkováva užívateľské rozhranie. To však musí komunikovať s procesom `Web_cfg`, ktorý inštruuje proces `Stat_flt_ctl`, ktorý riadi zápis alebo čítanie z interných registrov hardvérovej komponenty Filter.



Obr. 6.4: Architektúra softvérovej časti systému

Vstupné rozhrania tvoria komponenty `10G MAC` komunikujúce s hardvérovou časťou, označenou `FPGA`. Cez tieto 10 Gb rozhrania do softvéru prichádzajú metadáta zo zreťazených liniek implementovaných v hardvéry. Cez výstupné rozhrania, reprezentované taktiež komponentami 10G MAC, sú na kolektor exportované záznami o tokoch. Medzi vstupno-výstupné rozhrania softvérovej časti radíme `QSPI` a `1G MAC`, ktoré sa využívajú pri mapovanom prístupe do pamäti hardvérových komponent a manažmente sondy.

Monitorovanie a export tokov

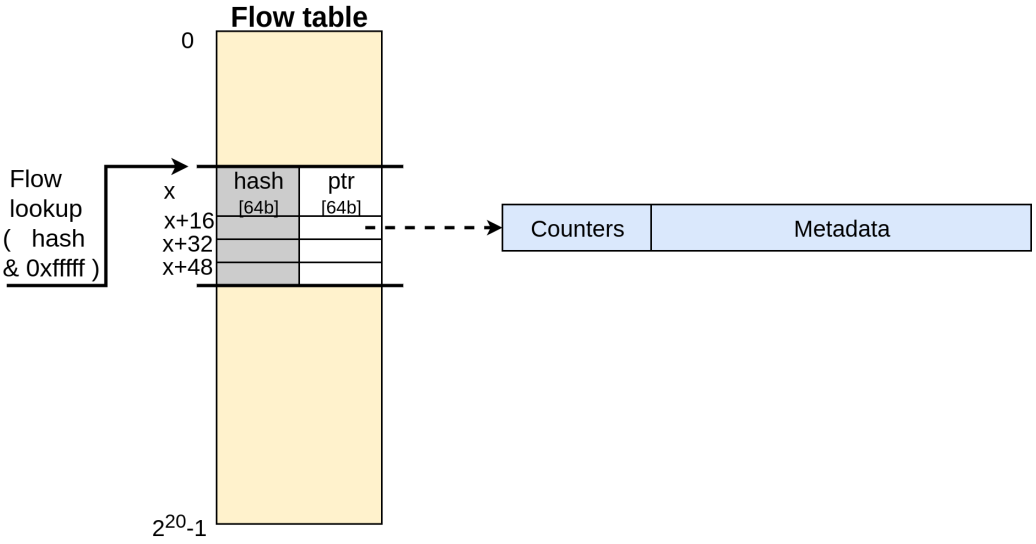
Monitorovanie tokov a export záznamov, sú implementované v rámci jedného procesu. Ten zodpovedá za načítavanie vstupných metadát, ich spracovávanie a následný export. Diagram tried softvérovej časti, ktorý je súčasťou prílohy A, znázorňuje začlenenie exportu v rámci monitorovania tokov. Táto úloha je implementovaná konkrétne triedou `IPFIX_exporter`, ktorá zabezpečuje vytvorenie TCP spojenia s kolektorom, správu užívateľom definovaných šablón a vytváranie IPFIX rámcov, ktorých formát odpovedá špecifikácií [11].

Základom implementácie monitorovania tokov je dátová štruktúra, v ktorej sú dočasne uložené záznamy o prebiehajúcich tokoch. S každým paketom spracovaným v hardvérovej časti je vygenerovaný jeden blok metadát. Z toho vyplýva, že softvérová časť bude musieť vykonávať postupnosť operácií s každým vstupným paketom. Týmito operáciami sú vyhľadanie, aktualizácia existujúceho alebo vloženie nového záznamu a kontrola časovačov.

Keďže vyhľadávanie medzi záznamami je najčastejšie vykonávanou operáciou, musíme vybrať vhodnú dátovú štruktúru, ktorá je optimalizovaná práve na tento typ operácie. Tok je v rámci nášho monitorovacieho systému určený hodnotami základných metadát, definovanými v podkapitole 6.1.1. Spolu tvoria zložený kľúč, na základe ktorého je vyhľadávanie vykonávané. Preto som zvolil **hash tabuľku**, ktorá umožňuje vyhľadávať v konštantnom čase. Využíva sa pri tom jednocestná funkcia, angl. *hash function*. Táto funkcia vytvorí zo zloženého kľúča celočíselnú hodnotu, ktorá slúži ako index poľa v tabuľke, na ktorom sa nachádza odpovedajúci záznam o toku. Ďalšou výhodou je, že hash je vypočítaný v hardvérovej časti a tak šetrí prácu procesora.

Záznam o toku obsahuje metadáta z prvého paketu, na základe ktorého bol nový tok vytvorený. Okrem nich obsahuje aj počítadlá, ktoré sú inkrementované s každou operáciou aktualizácie konkrétneho toku. Tieto počítadlá zaznamenávajú počet paketov patriacich do toku, ich celkovú veľkosť a časovú značku posledného paketu. Po aktualizácií toku na-

sleduje kontrola aktívneho time-outu, ktorý stanovuje maximálnu dĺžku toku. V prípade, že rozdiel časovej značky posledného a prvého paketu je väčší ako nakonfigurovaná hodnota časovaču, bude záznam o toku vyexportovaný a z tabuľky tokov odstránený.



Obr. 6.5: Schéma implementácie tabuľky tokov

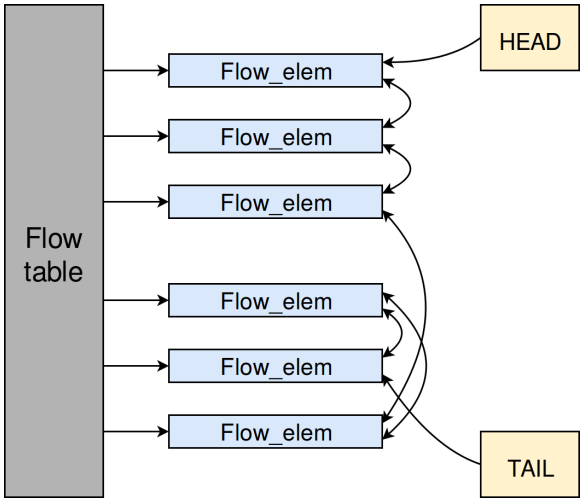
Tabuľka tokov je implementovaná formou jednorozmerného poľa. Na obrázku 6.5 môžeme vidieť, že každá položka poľa obsahuje dve 8 bajtové hodnoty. Prvou je odtlačok toku, `hash` a druhou je ukazovateľ na záznam o toku, `ptr`. Veľkosť tabuľky tokov bola stanovená na 1 048 576 položiek. Tým pádom nám na indexovanie tabuľky stačí 20 bitov. Preto je pri operácií vyhľadania toku, na obrázku označenej ako `Flow lookup`, použitých spodných 20 bitov z hashu, ktoré získame bitovou operáciou AND s konštantou `0xfffff`. Veľkosť tabuľky tokov bola stanovená nielen s ohľadom na spotrebu operačnej pamäti, ale aj s cieľom zmestiť čo najväčšiu časť tabuľky do pamäti cache. Pri súčasnej konfigurácii samotná tabuľka zaberá 16 MB, pri úplnom zaplnení tabuľky môže spotreba pamäte narásť až na 10 násobok tejto hodnoty. Pretože softvérová časť obsahuje 2 inštancie monitorovania tokov, je spotreba pamäti dvojnásobná.

Redukcia stavového priestoru indexovateľných tokov, znížením efektívnej šírky hashu na 20 bitov výrazne zvyšuje pravdepodobnosť **narodeninového paradoxu**, tzn. mapovania rôznych tokov na rovnaký index v tabuľke. Túto udalosť môžeme identifikovať dodatočným porovnaním celej 64 bitovej hodnoty hash z tabuľky tokov, s hodnotou hash, ktorá je súčasťou metadát získaných z hardvérovej časti sondy. V prípade nerovnosti týchto dvoch hodnôt, môžeme pokračovať v porovnávaní s nasledujúcimi položkami tabuľky. Tento prístup je označovaný ako **hashovanie so zásobníkom**, angl. *Bucket hashing* a využíva sa na znižovanie pravdepodobnosti kolízií a zvýšenia zaplnenia hashovacej tabuľky.

Pri správnej implementácii je možné využiť princíp lokality a dodatočné porovnania vykonať veľmi efektívne. Po prístupe k prvej položke sú do **rýchlej pamäte cache** uložené aj ďalšie položky zásobníka a preto je prístup k nim veľmi rýchly. Táto situácia je znázornená aj na obrázku 6.5, kde zásobník zarovnaný na šírku vyrovnávacej pamäte, angl. *Cache line*, je ohraničený vodorovnými čiarami. Na základe dokumentácie [1] spoločnosti ARM vieme, že procesor Cortex-A72, ktorý sa nachádza na platforme, využíva 64 bitové ukazovatele a vyrovnávaciu pamäť širokú 64 bajtov. Preto zásobník môže obsahovať až 4 položky, s veľ-

kosťou 16 bajtov, uložené v rámci jednej cache line. Z dôvodu tejto optimalizácie tabuľka tokov obsahuje iba dve najčastejšie využívané položky a nie všetky dáta.

Z popisu fungovania monitorovania tokov v podkapitole 2.3.1 vieme, že záznamy o tokoch sú exportované zo sondy v prípade vypršania aktívneho alebo neaktívneho time-outu. Z rozdielnosti týchto mechanizmov vyplýva, aj odlišnosť prístupu k ich implementácií. Na rozdiel od aktívneho, implementácia neaktívneho časovaču vyžaduje špeciálny mechanizmus kontrolujúci všetky toky a nie len tie, ktoré boli práve aktualizované. Tento mechanizmus je v diagrame tried, ktorý je súčasťou prílohy A, označený ako `TimeoutList`. Jedná sa o abstraktnú dátovú štruktúru obojsmerne zreťazeného zoznamu, ktorý spája všetky položky tabuľky tokov. Takto prepojené položky tvoria postupnosť usporiadanú podľa aktuálnosti toku. Aktuálnosť je vyjadrená časom príchodu posledného paketu patriacemu každému toku. Pri vytvorení novej alebo aktualizácií existujúcej položky v tabuľke tokov, je daná položka zaradená na začiatok obojsmerne zreťazeného zoznamu. Tým pádom zoradený zoznam obsahuje najaktuálnejšie toky na začiatku. Pri overovaní neaktívneho časovaču, nemusíme prechádzať všetky položky v tabuľke tokov. Stačí od konca prechádzať zreťazený zoznam a exportovať všetky položky pokiaľ nenájdeme tok, ktorého neaktívny časovač ešte nevypršal. Keďže je zoznam zoradený, je zaručené, že budú vyexportované všetky toky, ktorých neaktívny časovač vypršal. Vypršaný časovač je identifikovaný na základe porovnania času príchodu posledného paketu s aktuálnym časom. Ak rozdiel týchto dvoch hodnôt prekročí nakonfigurovaný limit, tak je záznam o toku vyexportovaný a z tabuľky tokov odstránený.



Obr. 6.6: Schéma obojsmerne zreťazeného zoznamu implementovaného nad tabuľkou tokov

Z dôvodu zachovania konzistencia dát a znižovania pamäťovej náročnosti softvérovej časti je vhodné implementovať obojsmerne zreťazený zoznam nad tabuľkou tokov. Na obrázku 6.6 je možné vidieť názorný príklad prepojenia položiek tabuľky tokov. Tá je znázornená šedým obdĺžnikom označeným `Flow table`. Každá položka tabuľky, na obrázku pomenovaná `Flow_elem`, obsahuje okrem dát aj ukazovateľ na svojho predchodcu a nasledovníka v rámci zoznamu. Prepojenie pomocou ukazovateľov je znázornené šípkami. Z dôvodu prehľadnosti obrázku sú dvojice jednosmerných šípok nahradené obojsmernými. Žltým obdĺžnikom je znázornený začiatok, `HEAD` a koniec zoznamu, `TAIL`.

Vzdialená konfigurácia

Možnosť ovplyvňovať funkcionalitu monitorovacieho systému počas jeho behu je súčasťou požiadavky na flexibilitu. Konfigurovať je možné komponentu filter, ktorá je implementovaná v hardvérovej časti sondy. Tým pádom je splnená aj požiadavka filtrovania paketov, ktorej súčasťou je možnosť zmeny filtrovacieho pravidla. Vzdialená konfigurácia filtra je pre užívateľa sprístupnená grafickým užívateľským rozhraním v časti **Hardvérový filter**, podkapitola 7.5.

Zo schémy zapojenia monitorovacieho systému, znázornenej na obrázku 6.1, je zjavné, že počítač sprostredkujúci GUI, komunikuje výhradne s úložiskom dát, ktoré odpovedá na jeho žiadosti o dáta. Keďže úložisko dát komunikuje so sondou, bude v kontexte vzdialenej konfigurácie v roli prostredníka, ktorý sprostredkováva komunikáciu medzi sondou a počítačom.

Súčasťou architektúry softvérovej časti, zobrazenej na obrázku 6.4, je proces `Web_cfg`, ktorý prijíma žiadosti odoslané užívateľským rozhraním. Tie sú spracované a vhodne mapované na príkazy konfiguračného programu `stat_flt_ctl`. Komponenta Filter, obsahuje pole registrov, ktoré reprezentujú hľadanú signatúru. Hodnoty v týchto registroch sú porovnávané so signatúrou spracovávaného paketu. Proces konfigurácie hardvérového filtra pozostáva z postupnosti čítaní a zápisov do jednotlivých registrov tejto komponenty.

6.2 Kolektor

Cieľom kolektoru je perzistentne ukladať dáta vygenerované monitorovacím procesom na sonde. Z hľadiska flexibility a univerzálnosti celého monitorovacieho systému je vhodné použiť štandardizovaný spôsob prenášania a ukladania informácii o sieťovej prevádzke, ako napr. NetFlow alebo IPFIX. Primárnou úlohou kolektoru je prijímanie a ukladanie rámcov, ktoré obsahujú záznamy o tokoch. Preto najdôležitejšou požiadavkou na zdroje zariadenia je dostatok diskového priestoru.

Z dôvodu prenosu aplikačne špecifických metadát, napr. signatúry, je potrebné využiť **protokol IPFIX**. Ako sa v sekcii 2.4 uvádza, tento protokol umožňuje definovať formát vlastných polí. Navyše, implementácia IPFIX kolektoru **IPFIXcol**, vytvorená združením **CESNET** je voľne dostupná spolu s rozsiahlou dokumentáciou [6]. Dobre dokumentovaná je hlavne časť popisujúca možnosti rozširovania tohto programu prostredníctvom prídavných modulov, čo je dôležité najmä z dôvodu ukladania stromu protokolov.

Na obrázku B.1 sú znázornené vstupné a výstupné moduly, ktoré IPFIXcol natívne podporuje. Tieto moduly definujú formát vstupných a výstupných dát. V rámci systému sú vstupné dáta preložené do internej reprezentácie a pred ich uložením môžu byť modifikované prostredníctvom prechodových modulov, na obrázku označených `intermediate plugins`, ktoré môžu napr. spájať, filtrovať alebo anonymizovať prijaté záznamy o tokoch. Preklad do internej reprezentácie je uskutočnený pomocou modulu správy šablón, na obrázku označenom `Template Manager`.

Jednotlivé moduly je možné v konfiguračnom súbore povoliť a navzájom prepojiť. Základom konfigurácie, ktorú som vytvoril pre nasadenie monitorovacieho systému je vstupný modul TCP počúvajúci na konkrétnej IP adrese, ktorá môže byť podľa potreby zmenená a porte 4739. Na vstupný modul je pripojený výstupný modul, **lnfstore**, ktorý ukladá záznamy o tokoch vo forme **NfDump** súborov. Ukladané súbory sú komprimované algoritmom **Lempel–Ziv–Oberhumer**. Štruktúra adresárov, do ktorých sú ukladané súbory

pozostáva z 3 úrovní zanorenia, ktoré postupne kódujú rok, mesiac, deň, pričom jeden súbor reprezentuje časové okno s dĺžkou 300 sekúnd.

S každým uloženým súborom vo formáte NfDump identifikovateľných podľa prefixu "`lnf.`", je vytvorený aj súbor s prefixom "`tree.`". V týchto súboroch je uložený strom protokolov odpovedajúci tokom uloženým v rámci daného časového okna. Táto funkcionalita bola presunutá zo sondy na kolektor s cieľom zvýšiť priepustnosť softvérovej časti presunutím výpočtov, ktoré nemusia byť nutne vykonávané sondou, do ostatných častí monitorovacieho systému. Keďže exportované záznamy o tokoch obsahujú signatúru toku, identifikátor vstupného rozhrania a oba VLAN tagy, je možné pomocou týchto informácií vybudovať strom protokolov aj na kolektore. Navyše odpadá réžia prenášania dynamických štruktúr protokolom IPFIX zo sondy na kolektor.

```
{
    "type": "object", "$id": "root",
    "properties": {
        "name": {"default": "all"},
        "packet_cnt": {"type": "number"},
        "byte_cnt": {"type": "number"},
        "flow_cnt": {"type": "number"},
        "children": {
            "type": "array", "$id": "node",
            "properties": {
                "name": {"type": "string"},
                "spec": {"type": "number"},
                "prot": {"type": "number"},
                "identifier": {"type": "number"},
                "children": {
                    "type": "array", "$ref": "#node"
                }
            }
        }
    }
}
```

Obr. 6.7: štruktúra stromu protokolov vo formáte JSON

Strom protokolov je ukladaný vo formáte **JSON**, ktorý je natívnym dátovým formátom jazyka **JavaScript**, v ktorom je implementované grafické užívateľské rozhranie zobrazujúce stromy protokolov. Na obrázku 6.7 je zobrazená štruktúra ukladaných objektov a ich atribúty. Koreň stromu, označený `root`, obsahuje počítadlá počtu paketov, bajtov a tokov. Jeho potomkovia, označení `node`, sú rekurzívne zanorení v atribúte typu pole, nazvanom `children`. Medzi ďalšie atribúty objektov node patrí `identifier`, ktorý jednoznačne identifikuje uzol v rámci stromu, `prot`, ktorý určuje protokol reprezentovaný daným uzlom a `spec`, ktorý rozlišuje inštancie rovnakého protokolu, napr. pre protokol IEEE 802.1Q obsahuje hodnotu VLAN tagu a tak umožňuje vetvenie stromu podľa VLAN.

Kapitola 7

Grafické užívateľské rozhranie

Cieľom grafického rozhrania, je čo najprehľadnejšie zobrazovať užívateľovi informácie. V oblasti sieťového monitoringu sú často využívané najmä grafy a tabuľky. Dôležité je tiež, aby práca s GUI bola pre užívateľa intuitívna a tak zvyšovala nielen produktivitu jeho práce, ale aj pravdepodobnosť, že na základe zobrazených informácií nájde hľadanú informáciu.

Logické usporiadanie užívateľského rozhrania musí odpovedať prípadom použitia systému a zároveň veličinám, ktoré vyplývajú z typu získavaných dát. V prípade monitorovania tokov a štruktúry protokolov sieťovej prevádzky, sú veličinami:

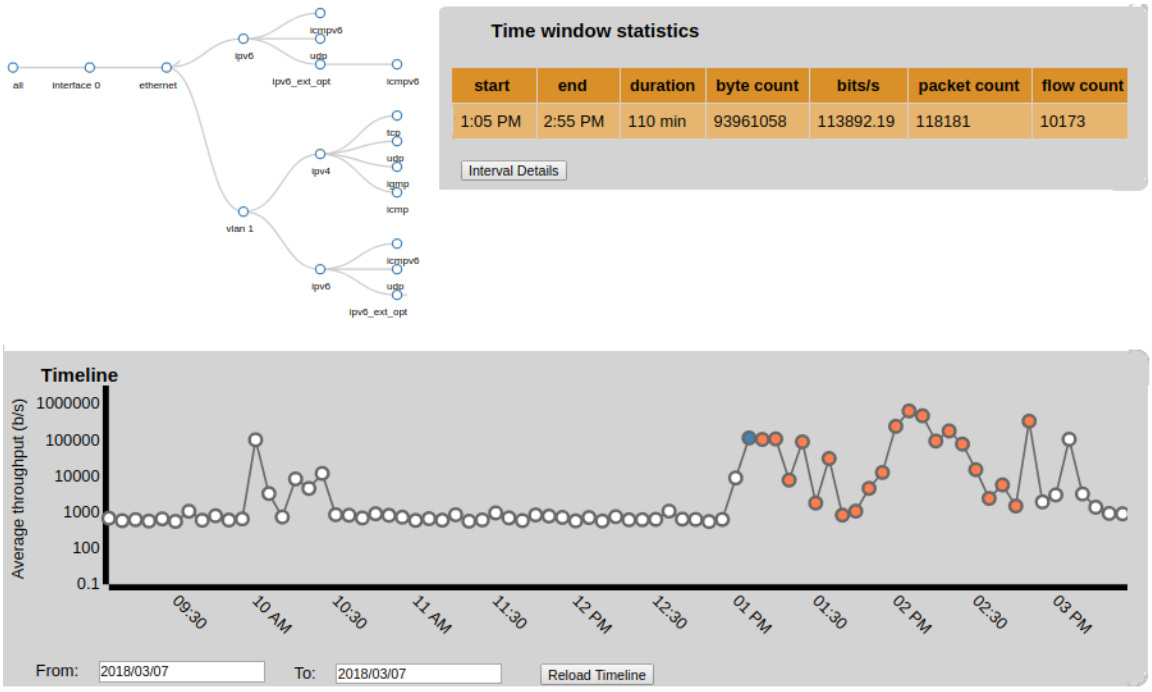
- **Čas** umožňuje vymedziť interval, v rámci ktorého sa nachádzajú informácie, ktoré chce užívateľ vizualizovať. Čím menší, je tento interval, tým presnejšie a relevantnejšie dáta užívateľ dostane. Množina záznamov, ktoré odpovedajú časovému intervalu, je stanovaná na základe časových značiek začiatku a konca toku.
- **Štruktúra zapúzdrenia hlavičiek** je zobrazovaná formou stromu protokolov. Jednotlivé uzly obsahujú agregované štatistiky, na základe ktorých si môže užívateľ vybrať konkrétnu signatúru, ktorá ho zaujíma.
- **Sieťové toky** je možné ďalej klasifikovať podľa hodnôt jednotlivých polí extrahovaných z hlavičiek protokolov. Záznam o toku je základnou jednotkou informácie, s ktorou v tomto monitorovacom systéme pracujeme.

Užívateľské rozhranie, ktoré som pre monitorovací systém navrhol a implementoval sa skladá práve z týchto troch pohľadov. Úlohou každého z nich je zobraziť sieťovú prevádzku z pohľadu danej veličiny, tak aby si užívateľ mohol vybrať jej podmnožinu. Tá bude znázornená užívateľovi v nasledujúcom pohľade. Preto môžeme tvrdiť, že každý z pohľadov je zároveň filtrom. Poradie, v akom nasledujú jednotlivé úrovne zobrazenia, reflektuje mieru detailu informácií, ktoré zobrazujú.

7.1 Časová os

Každý z pohľadov je vykreslený na samostatnej obrazovke. Prvá je vždy zobrazená užívateľovi časová os. Tento pohľad obsahuje najmenej detailov o sieťovej prevádzke. Ako môžeme na obrázku 7.1 vidieť, časová os je zobrazená ako lineárny graf veľkosti prevádzky v čase.

Užívateľ môže kliknutím na graf sieťovej prevádzky vybrať začiatok a koniec časového intervalu. V okne na pravej strane obrazovky s názvom `Time window statistics` sa aktualizujú informácie o vybranom časovom intervale. Konkrétne čas začiatku a konca, počet



Obr. 7.1: Časová os z prvej úrovne zobrazenia

paketov, bajtov, tokov a priemerná rýchlosť prenesených dát v rámci daného intervalu. Po kliknutí na tlačidlo `Interval details`, je užívateľovi zobrazená ďalšia úroveň detailu dát, ktoré patria do vybraného intervalu.

7.2 Štruktúra protokolov

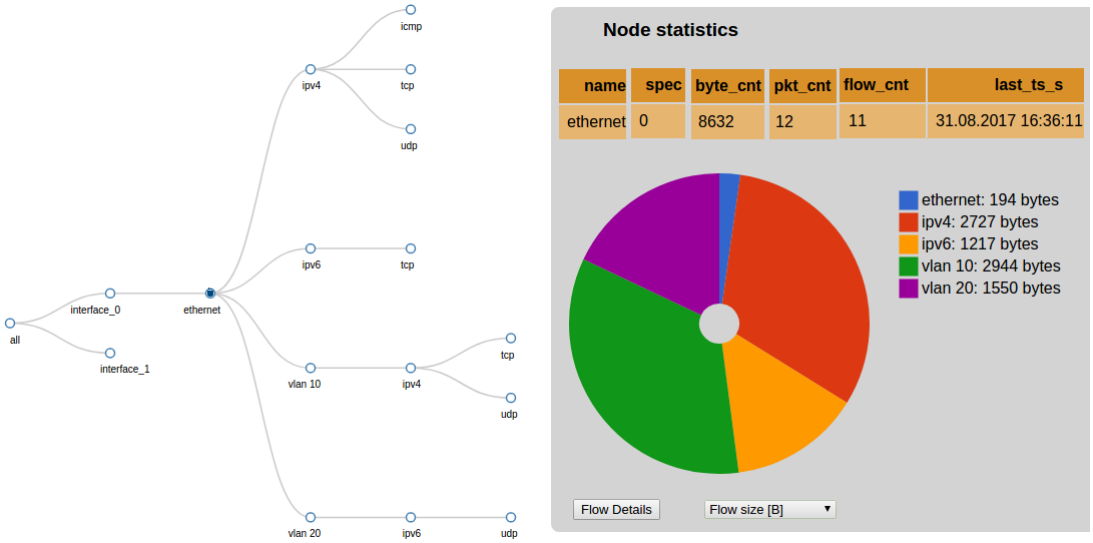
Druhá úroveň detailu zobrazuje štruktúru sieťovej prevádzky formou stromu protokolov. Tento strom je vytvorený podľa signatúr tokov, ktoré patria do vybraného časového intervalu. Ako môžeme na obrázku 7.2 vidieť, tento pohľad sa skladá zo stromu protokolov umiestneného vľavo a tabuľky s koláčovým grafom v pravej polovici obrázka.

Koreňom stromu protokolov je virtuálny uzol `All`, ktorý síce neodpovedá žiadnemu konkrétnemu protokolu, ale slúži na sumarizáciu celej prevádzky. Potomkami tohoto uzlu sú ďalšie virtuálne uzly, ktoré odpovedajú jednotlivým vstupným rozhraniam monitorovacieho systému. Ich podstromy obsahujú nevirtuálne uzly, ktoré tvoria stromy protokolov budované samostatne pre každé z rozhraní.

Užívateľ môže kliknutím ľavého tlačidla myši na uzol v strom protokolov zobraziť štatistiky, ktoré súvisia s vybraným uzlom a jeho podstromom. Dvojklik na uzol skryje celý podstrom a nadradený uzol zmení farbu na tmavomodrú. Skryté uzly je možné zviditeľni kliknutím pravého tlačidla myši na nadradený tmavomodrý uzol. Skrývanie jednotlivých uzlov umožňuje zobrazovať štatistiky iba tých podstromov, ktoré sú z pohľadu skúmania sieťovej prevádzky zaujímavé. Skryté uzly a im patriace toky sú ignorované a preto je skrývanie uzlov formou filtrovania záznamov o tokoch na základe ich štruktúry.

V okne s názvom `Node statistics` sa aktualizujú informácie o vybranom uzle. Tabuľka obsahuje čas príchodu posledného paketu a údaje o počte bajtov, tokov a paketov, ktoré patria vybranému uzlu alebo jednému z jeho potomkov. Koláčový graf znázorňuje rozloženie

prevádzky, medzi priamych potomkov vybraného uzlu a jeho samotného. V menu nachádzajúcom sa pod grafom je možné vybrať, či sa graf vykresľuje na základe počtu paketov, bajtov alebo tokov. Kliknutím na tlačidlo `Flow details`, je užívateľovi zobrazená ďalšia úroveň detailu, ktorá znázorňuje vybraný uzol a jeho podstrom.



Obr. 7.2: Štruktúra sieťovej prevádzky a štatistiky vybraného uzlu

7.3 Sieťové toky

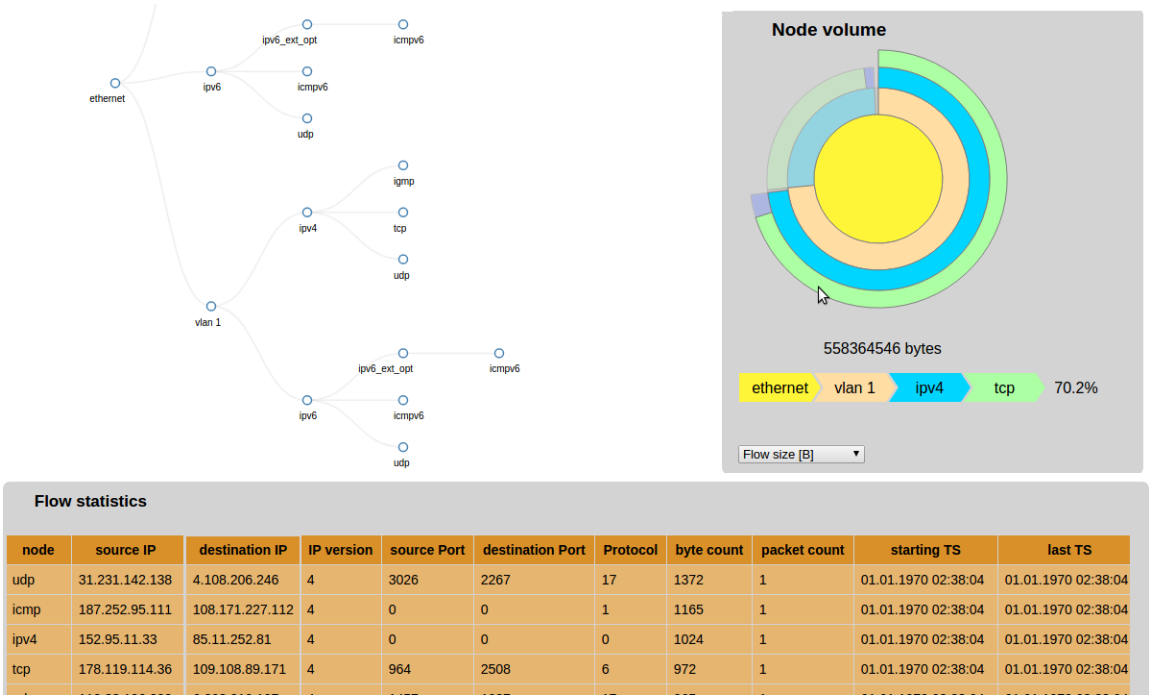
Posledný pohľad, poskytujúci najvyššiu úroveň detailu, zobrazuje záznamy o tokoch. Na obrázku 7.3 je možné vidieť, že tento pohľad obsahuje na ľavej strane vybraný uzol a jeho podstrom. Na pravej strane sa nachádza štruktúrovaný koláčový graf znázorňujúci rozloženie prevádzky, medzi všetkých potomkov vybraného uzlu. Po ukázaní kurzorom na časť grafu odpovedajúcemu niektorému z uzlov, sa pod grafom zobrazí jeho signatúra, hodnota a percentuálne zastúpenie v rámci celého podstromu. V menu nachádzajúcom sa pod grafom môže užívateľ vybrať, či sa graf vykresľuje na základe počtu paketov, bajtov alebo tokov.

V spodnej časti obrazovky sa zobrazí tabuľka s názvom `Flow statistics`. Obsahuje 30 najmohutnejších tokov z vybraného podstromu. Záznam o toku sa skladá z názvu uzlu, s ktorým je daný tok asociovaný, zdrojovej a cieľovej IP adresy, zdrojového a cieľového L4 portu, verzie IP protokolu, identifikátoru L4 protokolu, počtu paketov, bajtov a času začiatku a konca toku.

7.4 Softvérový filter

Okno s filtračným formulárom je súčasťou dvoch posledných náhľadov. Tento formulár umožňuje nastaviť, ktoré toky sú v štatistikách pre užívateľa zaujímavé a ktoré sa naopak majú ignorovať. Toto nastavenie ovplyvňuje iba dáta zobrazované užívateľovi prostredníctvom GUI.

Pokiaľ sú niektoré polia vo filtračnom formulári vyplnené, potom zobrazované štatistiky obsahujú iba toky, ktoré presne odpovedajú nastaveným parametrom. Prázdne pole vo formulári nemá vplyv na množinu vyfiltrovaných tokov.



node	source IP	destination IP	IP version	source Port	destination Port	Protocol	byte count	packet count	starting TS	last TS
udp	31.231.142.138	4.108.206.246	4	3026	2267	17	1372	1	01.01.1970 02:38:04	01.01.1970 02:38:04
icmp	187.252.95.111	108.171.227.112	4	0	0	1	1165	1	01.01.1970 02:38:04	01.01.1970 02:38:04
ipv4	152.95.11.33	85.11.252.81	4	0	0	0	1024	1	01.01.1970 02:38:04	01.01.1970 02:38:04
tcp	178.119.114.36	109.108.89.171	4	964	2508	6	972	1	01.01.1970 02:38:04	01.01.1970 02:38:04

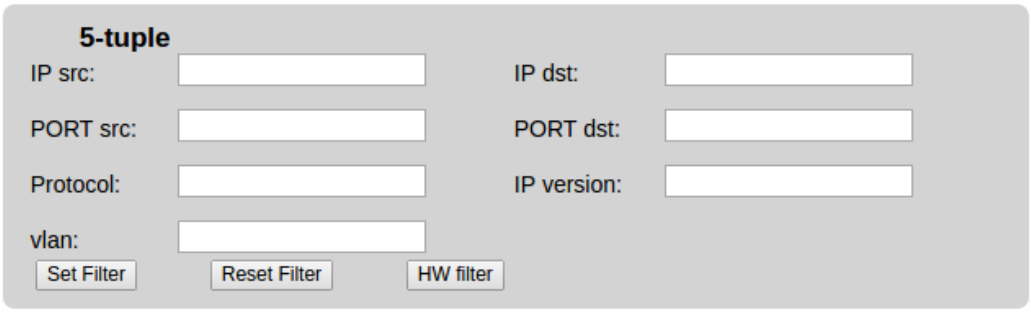
Obr. 7.3: Detail štruktúry vybraného podstromu spolu s asociovanými tokmi.

Na obrázku 7.4 môžeme vidieť, že filtračný formulár umožňuje filtrovať štatistické dáta na základe zdrojovej a cieľovej IP adresy, zdrojového a cieľového L4 portu, verzie IP protokolu, čísla L4 protokolu a VLAN identifikátoru.

Pod filtračným formulárom sa nachádzajú tlačidlá `Set filter`, ktoré aplikuje pravidlo zadané do filtračného formulára, `Reset filter`, ktoré zresetuje filter a zobrazí všetky štatistiky a `HW filter`, ktoré prejde ku konfigurácii hardvérového filtru.

7.5 Hardvérový filter

Samostatné zobrazenie, obsahujúce formulár, prostredníctvom ktorého môže užívateľ konfigurovať komponentu Filter implementovanú v FPGA, sa v rámci GUI nazýva Hardvérový filter. Tento filter rozhoduje na základe sekvencie zapúzdrených hlavičiek v pakete, či bude paket z hardvérovej časti zahodený alebo exportovaný.

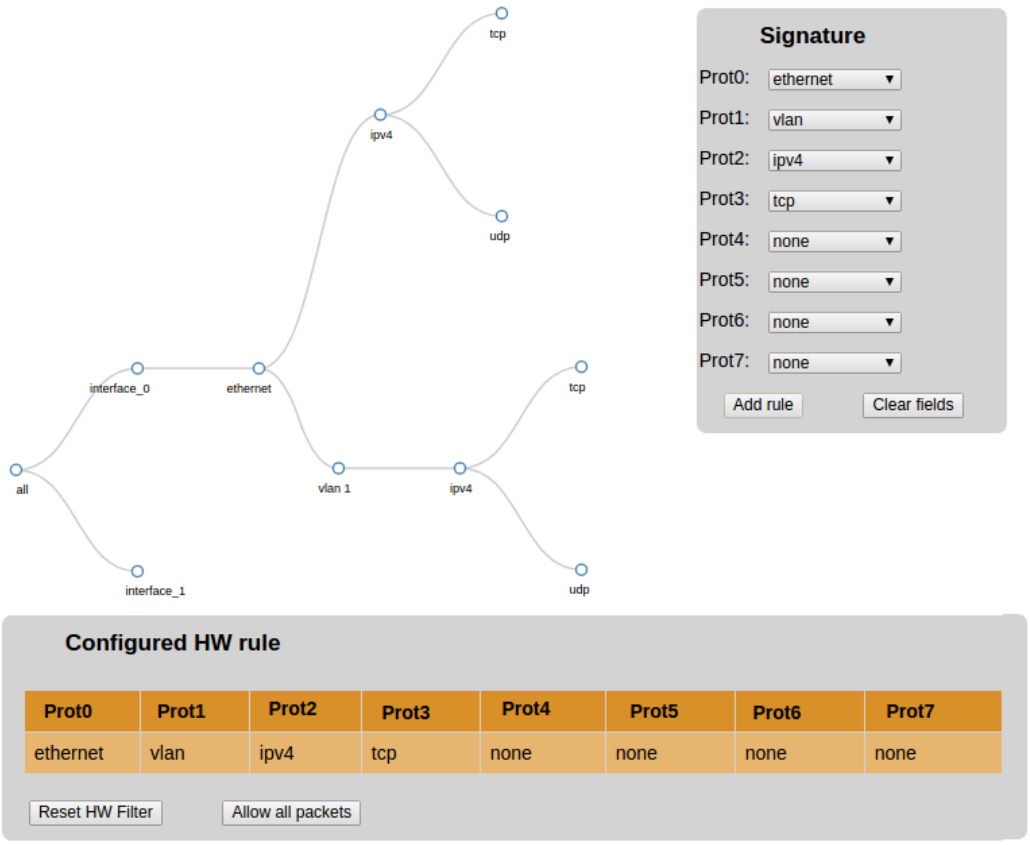


Obr. 7.4: Filtračný formulár

Na obrázku 7.5 môžeme vidieť, že okno s názvom **Signature** obsahuje 8 polí, `Prot 0` až `Prot 7`, reprezentujúcich požadovanú postupnosť protokolov. V každom z týchto polí užívateľ môže vybrať názov jedného z podporovaných protokolov alebo jednu z dvoch špeciálnych hodnôt. Prvou z týchto hodnôt je `allow`, ktorá povoľuje výskyt všetkých protokolov a tým umožňuje vytvárať masky odpovedajúce viacerím signatúram. Druhou špeciálnou hodnotou je `None`, ktorá nepripúšťa žiadny protokol a slúži na obmedzenie počtu zapuzdrených hlavičiek. Pravidlo je nakonfigurované po kliknutí na tlačidlo `Add rule`.

Strom protokolov, ktorý je zobrazený v ľavej časti obrazovky, môže byť taktiež využitý pri vypĺňaní signatúry. Po kliknutí na niektorý z jeho uzlov, sa polia samé vyplnia signatúrou daného uzlu.

V spodnej časti obrazovky je tabuľka, ktorá obsahuje aktuálnu konfiguráciu hardvérového filtra. Pod ňou sa nachádzajú dve tlačidlá, `Allow all packets`, ktoré nakonfiguruje filter tak, aby prepúšťal všetky pakety. Tlačidlo `Reset HW filter` nastaví filter do východzej konfigurácie, kedy sú všetky pakety zahadzované.



Obr. 7.5: Časť GUI určená ku konfigurácii hardvérového filtra

Kapitola 8

Testovanie

Overenie správnosti implementácie vzhľadom na špecifikáciu systému je veľmi dôležité. Keďže sa jedná o heterogénny systém, pre každú z technológii, ktoré boli použité, je potrebné zvoliť iný prístup k testovaniu. Zároveň je nutné overiť správnosť komunikácie, medzi jednotlivými výpočtovými časťami. Z tohto dôvodu som vytvoril dve nezávislé testovacie prostredia, ktoré vychádzajú z týchto testovacích metód:

- **Funkčná verifikácia** číslicových systémov používa pri overovaní správnosti systému nielen simuláciu, ale aj iné pokročilé techniky ako: generovanie pseudo-náhodných stimulov, samokontrolné mechanizmy, kontrolu formálnych tvrdení a verifikáciu riadenú pokrytím [12].

  Na základe špecifikácie je vytvorený model obvodu. Na vstup modelu a verifikovaného obvodu, angl. *design under test*, skr. **DUT**, sú privedené rovnaké hodnoty a ich výstupy sú porovnané na zhodu. Funkčná verifikácia je ukončená, keď dosiahneme požadovanú úroveň pokrytia funkčnosti.

  Pri tvorbe verifikačného prostredia a modelu som použil jazyk **System Verilog** a metodológiu **UVM**, z angl. *Universal Verification Methodology*, ktorá je štandardom pri verifikovaní číslicových obvodov.

- **Automatizované testy** sa skladajú z testovacieho prostredia a sady testov. Testovacie prostredie riadi postupne spúšťanie testov ako aj overovanie správnosti výstupov. Jednotlivé testy reflektujú funkčné požiadavky testovaného systému.

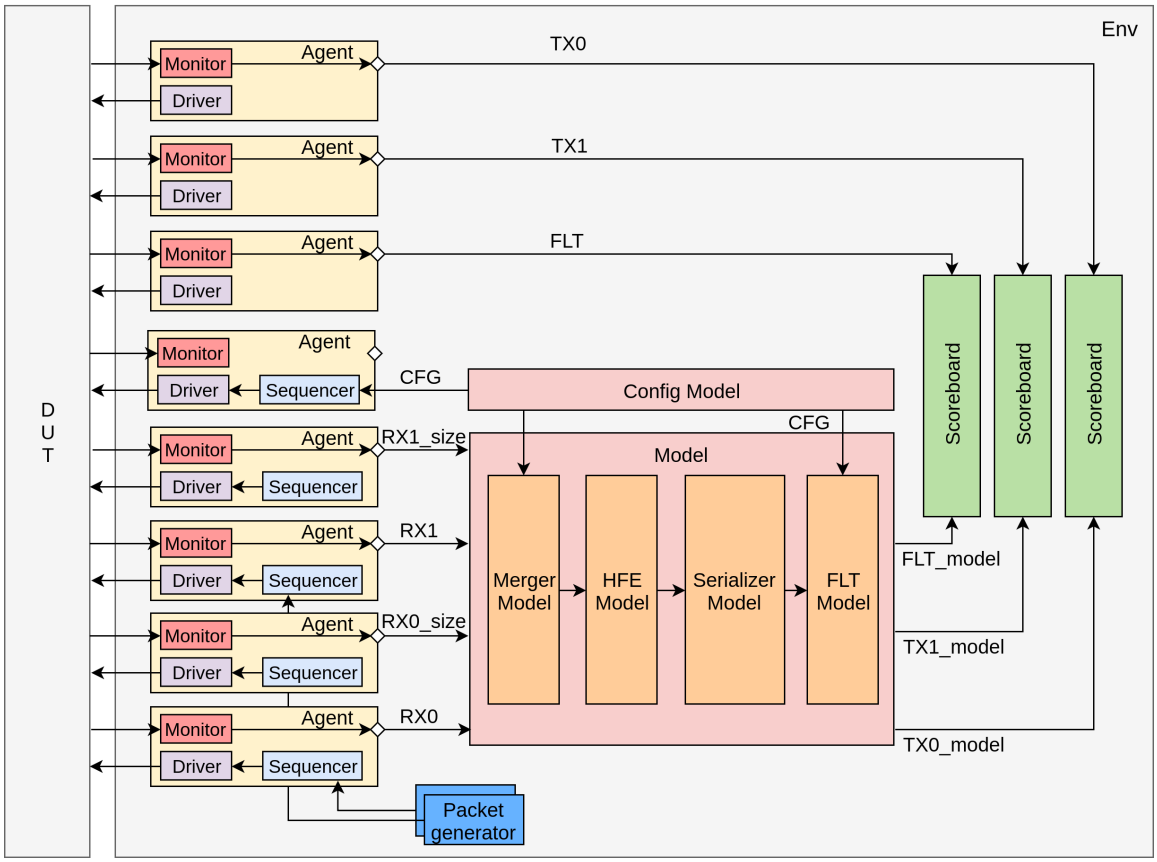
  Tento prístup k testovaniu umožňuje overiť funkčnosť monitorovacieho systému ako celku a je schopný identifikovať chyby nielen v softvérovej a hardvérovej časti, ale aj v komunikácií jednotlivých modulov.

8.1 Funkčná verifikácia hardvérovej časti

Pri funkčnej verifikácií hardvérovej časti monitorovacieho systému som využíval metodológiu **UVM**, ktorá reprezentuje verifikačné prostredie ako súbor objektov, ktoré rozdeľujeme na **komponenty** a **transakcie**. Komponenty tvoria hierarchickú štruktúru a medzi sebou komunikujú prostredníctvom transakcií.

Verifikačné prostredie sa skladá z `generátora paketov` [22], ktorý generuje dáta pre vstupnú transakciu vznikajúcu v komponente `Sequencer`. Všetky vstupné transakcie sú privedené na vstup `DUT` a referenčného modelu prostredníctvom `Agentov`. Výstupy sú vedené

z DUT cez agentov, ktorí z nich vytvárajú výstupné transakcie, ktoré sú v komponentách `Scoreboard` porovnávané s odpovedajúcimi výstupnými transakciami z modelu.



Obr. 8.1: Schéma verifikačného prostredia vytvoreného podľa metodológie UVM

Na obrázku 8.1 je možné vidieť, že `Model`, ktorý reprezentuje funkcionalitu verifikovaného obvodu, pozostáva z niekoľkých submodelov, ktoré odpovedajú komponentám zreťazenej linky implementovanej v hardvérovej časti. Komponenta Model vyberá dvojice vstupných transakcií `RX0`, `RX0_size` a `RX1`, `RX1_size`, ktoré obsahujú pakety a ich veľkosti. Odpovedajúce výstupné transakcie, ktoré sú na obrázku 8.1 označené `TX0_model` a `TX1_model` obsahujú získané metadáta z paketov a sú porovnávané s transakciami `TX0` a `TX1` v uvedenom poradí. Posledná dvojica výstupných transakcií, `FLT` a `FLT_model`, obsahuje pakety, ktoré sú exportované filtrom na základe ich signatúry.

Funkcionalita niektorých komponent zreťazenej linky sa mení na základe ich vnútornej konfigurácie. Verifikačné prostredie, preto obsahuje komponentu `Config Model`, ktorá konfiguruje komponenty DUT a odpovedajúce submodely, pomocou rozhrania `CFG`, tak aby bolo ich chovanie rovnaké a funkčná verifikácia pokrývala aj stavový priestor rôznych konfigurácií hardvérových komponent.

Verifikačné prostredie obsahuje 2 testy, ktoré sa navzájom líšia konfiguráciou filtra a teda očakávanými výstupnými transakciami rozhrania FLT a FLT_model. Každý z testov generuje 100 miliónov paketov rôznej štruktúry, veľkosti a hodnôt extrahovaných položiek. Pri procese verifikácie bolo odhalených hneď niekoľko chýb, najmä v prepojení a komunikácií niektorých komponent zreťazenej linky, endianity extrahovaných dát a formátu serializácie.

Nakoniec však verifikovaný obvod úspešne prešiel všetkými testami, pričom pokrytie kódu dosiahlo 95 %.

8.2 Automatizované testy monitorovacieho systému

Automatizované testy slúžia k testovaniu jednotlivých funkcií monitorovacieho systému a sú založené na metóde **black-box**. Funkcionalita monitorovacieho systému je skúmaná bez znalosti jeho interných štruktúr alebo spôsobu implementácie. Automatizované testy pozostávajú okrem testovacieho prostredia, aj zo sady predpripravených vstupných dát.

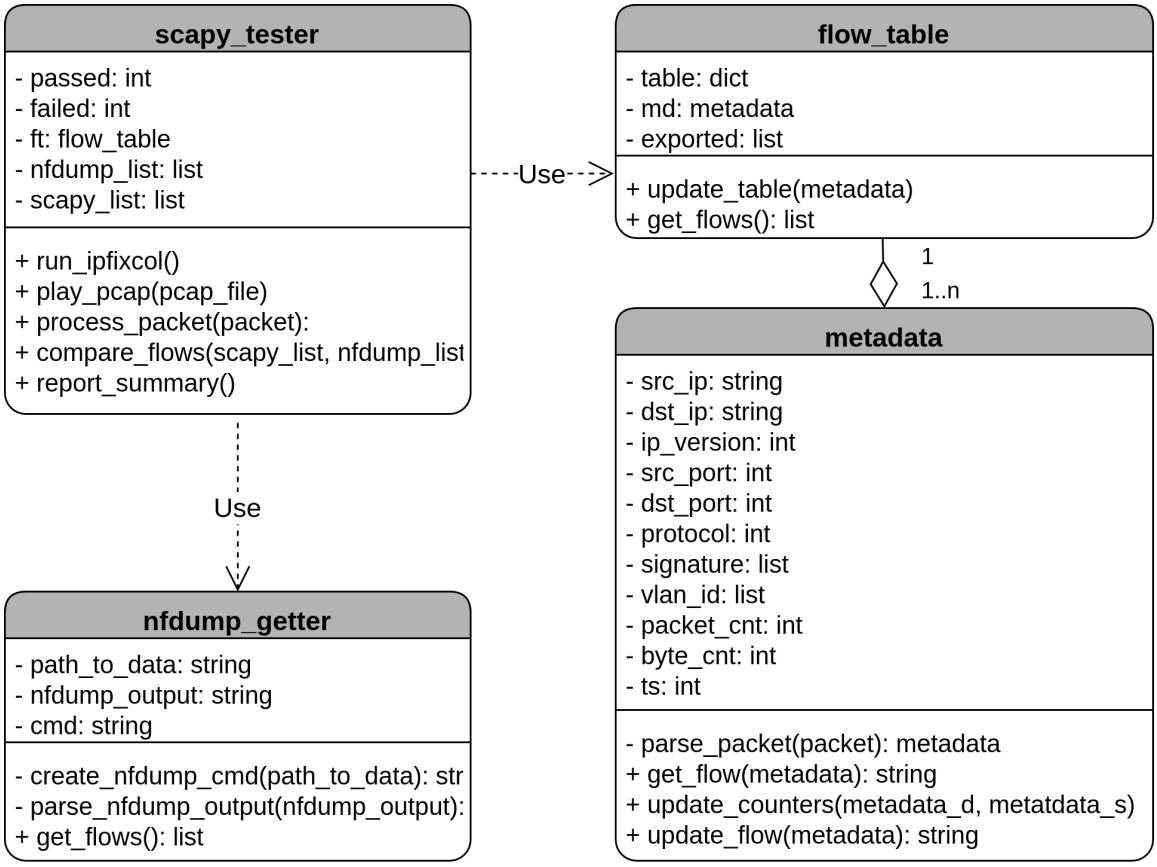
Úlohou testovacieho prostredia je riadiť postupné spúšťanie testov, ukladanie výstupov monitorovacieho systému, počítanie referenčných výstupov, ich zmysluplné porovnávanie a vyhodnocovanie úspešnosti testov. Okrem vlastných tried testovacie prostredie využíva sadu open-source nástrojov: **tcpreplay**, ktorý slúži na prehrávanie paketov uložených v súbore, **ipfixcol**, ktorý je implementáciou ipfix kolektoru a **nfdump**, ktorý vyčítava informácie o tokoch uložených kolektorom. Samotné testovacie prostredie je implementované v jazyku **Python** verzie 3.5.2.

Na obrázku 8.2 je zobrazený diagram tried, ktorý obsahuje vzťahy medzi triedami, ako aj ich základné metódy a atribúty.

- **Trieda scapy_tester** riadi celý proces testovania. Postupne vyberá testovacie súbory, ktoré obsahujú vstupné dáta - **pakety**. Každý test je vykonaný ako postupnosť krokov: spustenie ipfix kolektoru, prehranie paketov na vstup sondy, výpočet referenčných výstupov, načítanie výstupov monitorovacieho systému a porovnanie výstupov.
- **Trieda metadata** zabezpečuje spracovávanie paketov načítavaných zo súboru a získavanie rovnakých metadát, aké poskytuje hardvérová časť monitorovacieho systému. Pri implementácii tejto triedy som využil knižnicu **scapy** vo verzii 2.4.0, ktorá implementuje načítavanie paketov a rozpoznávanie hlavičiek protokolov.
- **Trieda flow_table** zodpovedá za agregáciu metadát do tokov, spôsobom odpovedajúcim softvérovej časti monitorovacieho systému. Preto je potrebné, aby time-outy, ktoré riadia exportovanie tokov zo sondy, odpovedali hodnotám time-outov v tejto triede.
- **Trieda nfdump_getter** vytvára vhodnú žiadosť o dáta uložené ipfix kolektorom, ktoré získava prostredníctvom volania nástroja **nfdump**. Záznamy o tokoch ďalej formátuje do vhodnej podoby tak, aby ich bolo možné porovnať s referenčnými tokmi, ktoré sú spravované triedou flow_table.

Súčasťou automatizovaného testovania je aj množina testov, pričom každý test je reprezentovaný jedným súborom vo formáte **.pcap**. Testy sú zamerané na tri základné oblasti funkcionality monitorovacieho systému. Každá z týchto oblastí obsahuje 10 samostatných testov, ktoré obsahujú až tisíce paketov.

- **Testy extrakcie metadát** sú zamerané na overenie správnosti získaných metadát. Toky neobsahujú veľké množstvo paketov, ale vyznačujú sa rôznorodosťou zapúzdrenia protokolov ako napr. Q in Q, IP inIP alebo využitím rozširujúcich IPv6 hlavičiek.
- **Testy agregácie tokov** kontrolujú správnosť agregácie metadát a hodnoty počítadiel v rámci tokov. Toky môžu obsahovať väčšie množstvo paketov, avšak nie sú natoľko dlhé, aby dochádzalo k vypršaniu aktívneho alebo pasívneho časovaču.



Obr. 8.2: Diagram tried testovacieho prostredia pre automatizované testy

- **Testy funkcionality časovačov** obsahujú toky, v rámci ktorých sú pakety oddelené výraznejšími medzipaketovými medzerami, tak aby dĺžka spracovania paketov na sonde, bola zanedbateľná oproti rozostupom medzi paktami, ktoré určujú časové značky v pcap súbore. Tak zabránime vzniku odchýlok medzi testovaným systémom a modelom. Dôsledkom by bola nedeterministická príslušnosť niektorých paketov do záznamov o tokoch, ktoré boli rozdelené vypršaním aktívneho alebo neaktívneho časovaču, čo by spôsobilo chyby pri porovnávaní výstupov.

Kvôli nedostupnosti platformy, bola automatizovane testovaný variant monitorovacieho systému implementovaný na zariadení **Zynq4Ethernet** od spoločnosti **NetModule**. Z hľadiska návrhu je rozdiel medzi týmito dvoma platformami minimálny. Dekompozícia systému a mapovanie na výpočtové zdroje je totožné. Monitorovací proces softvérovej časti je spustený v jednej inštancii a hardvérová časť obsahuje jednu zreťazenú linku, ktorá spracováva pakety z oboch vstupných rozhraní s rýchlosťou 1 Gbps.

Podpora testov jednotlivých oblastí funkcionality bola implementovaná postupne. Počas testovania pomocou automatizovaného testovacieho prostredia boli odhalené chyby výpočtu veľkosti paketu a offsetu aplikačných dát. Tieto chyby sa vyskytovali v IP paketoch, ktoré boli pôvodne menšie ako 64 bajtov. V tom prípade bol na konci ethernetovej hlavičky doplnený potrebný počet bajtov, tak aby výsledný rámec spĺňal požiadavku na minimálnu veľkosť. Nakoniec všetkých 30 testov prebehlo úspešne.

Kapitola 9

Dosiahnuté výsledky

Po úspešnom overení funkcionality je potrebné zmerať nároky hardvérovej architektúry na zdroje a výkonnosť celého monitorovacieho systému. Na základe týchto meraní je možné vyhodnotiť úspešnosť návrhu, mapovania úloh na výpočtové zdroje a implementácie.

9.1 Spotreba zdrojov

Pod pojmom zdroje sú v tejto podkapitole označované základné typy stavebných blokov, z ktorých sa skladajú FPGA čipy od firmy Altera. Konkrétne sa jedná o 8-vstupé adaptívne logické moduly, označované **ALM**, ktoré implementujú rôzne kombinácie vyhľadávacích tabuliek a vstavané pamäťové bloky, **M20k**, ktorých spotreba je vyjadrená počtom potrebných bitov. Okrem toho získavame údaje o využití **DSP** blokov, ktoré podporujú rôzne aritmetické operácie a **registrov**, ktoré sa využívajú ako rýchle pamäte pre zlepšenie časovania obvodu.

V tabuľke 9.1 je uvedené množstvo zdrojov, ktoré spotrebujú jednotlivé komponenty hardvérovej časti sondy po fáze syntézy. Posledný riadok tabuľky obsahuje celkovú spotrebu všetkých komponent, s ohľadom na počet inštancií jednotlivých komponent. Všetky informácie o využití zdrojov sú získané z programu **Quartus Prime** vo verzii 17.1.1. Cieľovým zariadením je FPGA čip s označením **10AX027E1F27E1SG**.

component	ALM	register	block memory bits	DSP
data export	26	19	0	0
merger	159	89	6240	0
axi-lite ipif	9	9	0	0
aggregator	1864	538	584	0
binder	95	83	32	0
axis_fifo	44	25	2048	0
fork	10	134	0	0
filter	25	101	0	0
hash unit	1059	619	0	0
hfer	1600	1220	32768	0
serializer	284	19	5632	0
tsu	140	285	0	3
total	10359	5886	94576	3

Tabuľka 9.1: Spotreba zdrojov komponent hardvérovej časti po syntéze

Tabuľka 9.2 obsahuje údaje o celkovej spotrebe zdrojov po fáze **place & route**. K týmto hodnotám bolo vypočítané aj percentuálne zaplnenie čipu. Keďže registre sú súčasťou ALM, tabuľka neobsahuje samostatný údaj o ich percentuálnej spotrebe.

	ALM		register	block memory		DSP	
	počet	%	počet	bit	%	počet	%
10G design	7648	8 %	5913	94576	<1 %	3	<1 %

Tabuľka 9.2: Celková spotreba zdrojov a percentúalne zaplnenie čipu

Z hodnôt, ktoré sú uvedené v predchádzajúcich dvoch tabuľkách vyplýva, že po optimalizáciách, ktoré sú vykonávané vo fáze place & route, klesla spotreba logiky implementovanej pomocou blokov ALM oproti hodnotám po syntéze o 25 %. Aj vďaka tomu dosiahla celková spotreba ALM iba 8 % a spotreba ostatných zdrojov dokonca menej ako 1 %. Vytvorený dizajn spĺňa časovanie a analýza časovania vypočítala maximálnu pracovnú frekvenciu obvodu až 198 MHz, čo je spolu s nízkou spotrebou zdrojov vynikajúci výsledok.

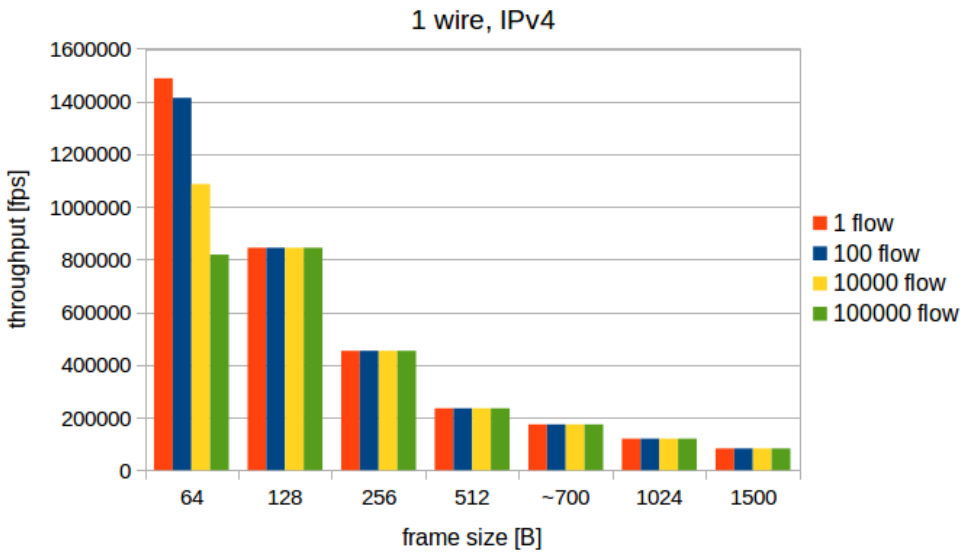
9.2 Priepustnosť systému

Z dôvodu spomínanej nedostupnosti platformy, bola priepustnosť systému meraná na zariadení **Zynq4Ethernet** od spoločnosti **NetModule**. Toto zariadenie obsahuje vstupné rozhrania s rýchlosťou 1Gbps, z čoho vyplýva že jeho maximálna priepustnosť je 10 násobne menšia, ako priepustnosť platformy, pre ktorú bol monitorovací systém navrhovaný a implementovaný. Keďže Zynq4Ethernet obsahuje procesor ARM Cortex-A9 a softvérová časť spúšťa iba jednu inštanciu procesu vykonávajúceho monitorovanie tokov, môžeme tvrdiť, že výkonnosť platformy obsahujúcej jadrá ARM Cortex-A72 s dvoma monitorovacími procesmi bude aspoň 10 násobne väčšia.

Všetky údaje boli namerané pomocou prístroja **Spirent TestCenter**, ktorý generoval prevádzku s rôznou **veľkosťou rámcov**, **štruktúrou protokolov**, odlišného počtu **tokov** a **vstupných rozhraní**. Každé meranie prebiehalo aspoň **60 sekúnd**, aby sa mohla prejaviť réžia exportu tokov a kontroly časovačov. Ak v danom časovom intervale došlo k zahodeniu paketov vstupným rozhraním sondy, objem generovanej prevádzky bol zmenšený a meranie bolo zopakované. V opačnom prípade sa objem prevádzky zvyšoval, pokiaľ nebola nájdená maximálna priepustnosť danej konfigurácie. Priepustnosť bola meraná v počte prenesených bajtov, **Bps** a rámcov, **fps**, za sekundu. Namerané hodnoty, na základe ktorých boli všetky grafy vytvorené, obsahujú tabuľky v prílohe C.

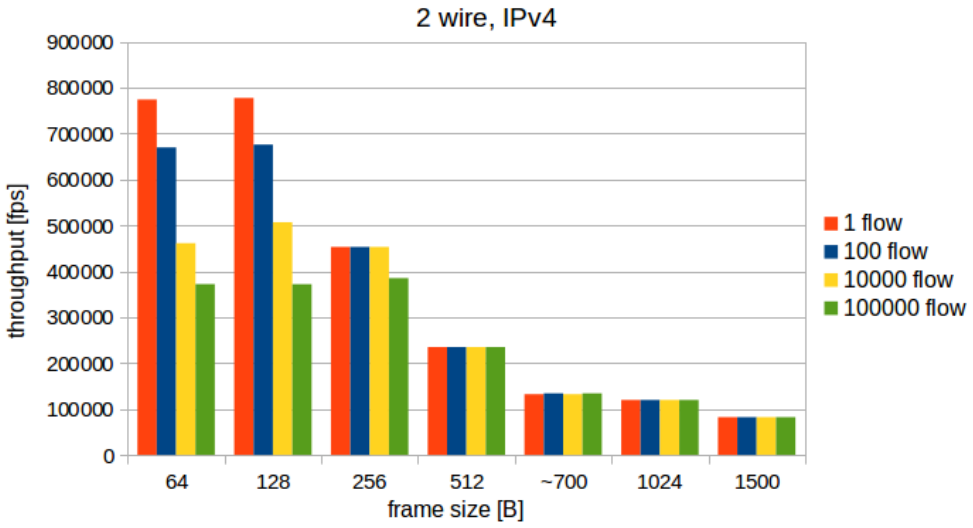
Nasledujúce štyri grafy zobrazujú závislosť priepustnosti meranej v rámcoch za sekundu na veľkosti rámcov a počte generovaných tokov. Pri meraniach som okrem fixnej veľkosti rámcov, používal aj generovanie veľkosti podľa rovnomerného rozdelenia ohraničeného rozsahom 100 a 1300 bajtov. Stredná hodnota je 700 bajtov, preto je v grafoch tento variant označený **∼ 700**. Každý z grafov zobrazuje výsledky meraní pre rôzne počty vstupných rozhraní a štruktúru rámca, ktorý pozostáva z ethernetovej a IPv4 alebo IPv6 hlavičky.

Na obrázku 9.1 je znázornený graf priepustnosti pri využití jedného vstupného rozhrania a rámcov s IPv4 hlavičkou. Pri generovaní jediného toku sonda stíha spracovávať všetky pakety na vstupe. So zvyšovaním počtu tokov priepustnosť pri najmenších rámcoch postupne klesá, pretože čoraz častejšie dochádza k výpadkom v pamäti cache. Tým pádom sa znižuje rýchlosť celého spracovania na procesore. Pri rámcoch väčšej veľkosti už k tomuto javu nedochádza, pretože metadáta z hardvérovej časti majú konštantnú veľkosť. Tým pádom je



Obr. 9.1: Graf priepustnosti konfigurácie s 1 vstupným rozhraním a protokolom IPv4

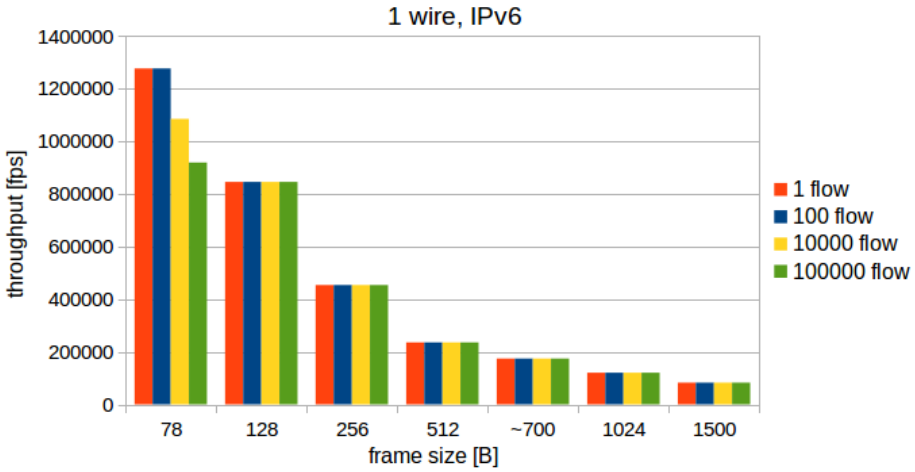
priepustnosť softvérovej časti závislá na počte spracovaných rámcov a nie objeme vstupnej prevádzky.



Obr. 9.2: Graf priepustnosti konfigurácie s 2 vstupnými rozhraniami a protokolom IPv4

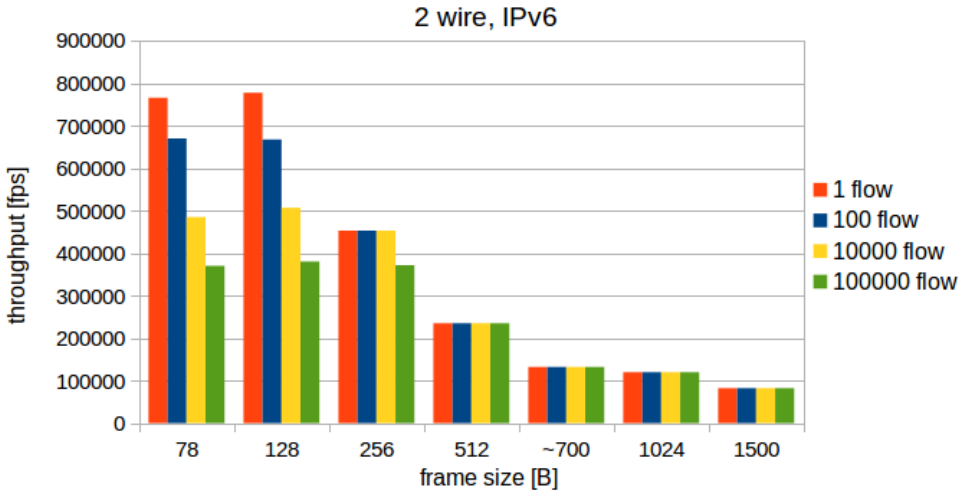
Obrázok 9.2 obsahuje graf meraní pre dva aktívne vstupy, ktoré majú rovnakú konfiguráciu. Graf zobrazuje priepustnosť jedného vstupného rozhrania, a teda celková priepustnosť je dvojnásobne väčšia. Môžeme si všimnúť, že namerané hodnoty pre rámce veľkosti 64 a 128 bajtov sú takmer rovnaké, čo podporuje predošlé tvrdenie o vplyve počtu spracovaných paketov. Je vidieť, že bola dosiahnutá maximálna celková priepustnosť zhruba 1500000 paketov za sekundu.

Graf na obrázku 9.3 zobrazuje merania pre jedno vstupné rozhranie a pakety obsahujúce na protokol IPv6. Keďže veľkosť jeho hlavičky je až 40 bajtov, je možné generovať



Obr. 9.3: Graf priepustnosti konfigurácie s 1 vstupným rozhraním a protokolom IPv6

rámce s minimálnou veľkosťou 78 bajtov. Rovnako ako pri predchádzajúcej konfigurácii s jedným vstupným rozhraním je linka vyťažená naplno, ešte pred dosiahnutím maximálnej priepustnosti, vo všetkých prípadoch okrem meraní s najmenšími rámcami.

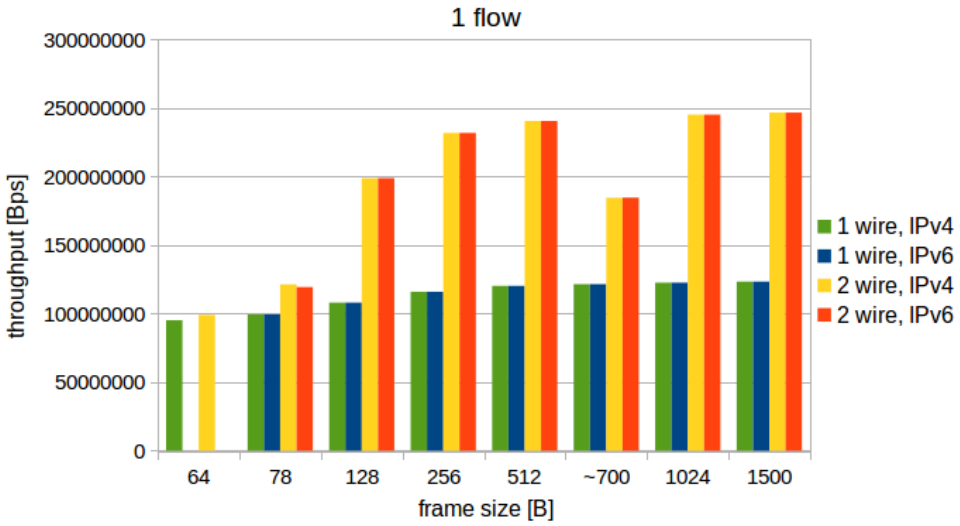


Obr. 9.4: Graf priepustnosti konfigurácie s 2 vstupnými rozhraniami a protokolom IPv6

V grafe priepustnosti konfigurácie s dvoma vstupnými rozhraniami a hlavičkami protokolu IPv6, zobrazenom na obrázku 9.4, si môžeme všimnúť malé, zhruba 1 %, rozdiely v priepustnosti medzi meraniami s rovnakým počtom tokov, pri veľkosti paketov 78 a 128 bajtov. Namerané hodnoty dosahujú maximálnu celkovú priepustnosť systému a preto by som odchýlky prisúdil chybe merania.

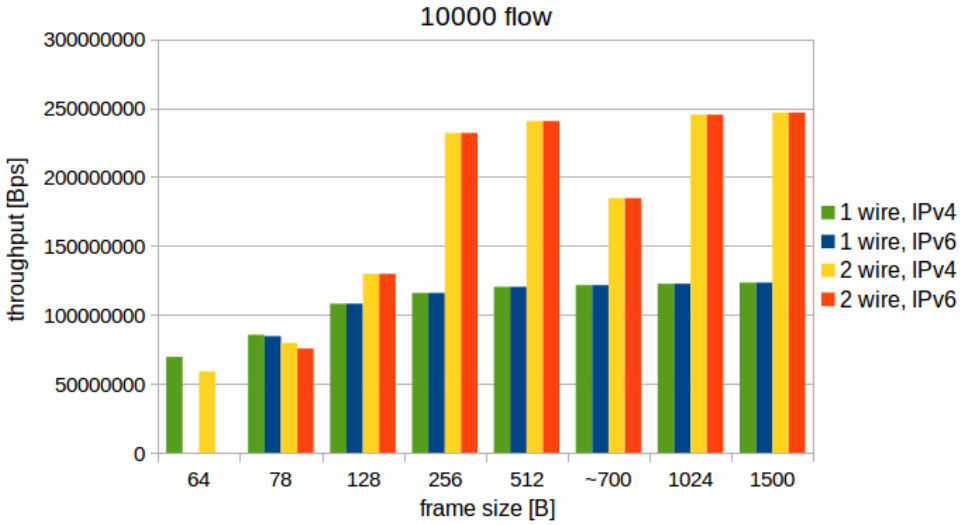
Posledné dva grafy znázorňujú porovnanie celkovej priepustnosti systému v štyroch základných konfiguráciách. Na rozdiel od predchádzajúcich grafov, ktoré boli zamerané na vizualizáciu maximálnej priepustnosti v jednotlivých konfiguráciách, tieto grafy porovnávajú celkovú priepustnosť a identifikujú odchýlky medzi konfiguráciami.

Na obrázku 9.5 je znázornený graf priepustnosti pre 1 tok. Graf obsahuje zaujímavý pokles nameranej priepustnosti pri rovnomernom rozdelení veľkosti rámcov. Keďže prie-



Obr. 9.5: Graf priepustnosti pre 1 tok

merná veľkosť rámcov pri tomto rozložení je 700 bajtov, sonda by mala podľa očakávaní bez problémov spracovávať prevádzku na plne vyťaženej linke. Pri dvoch vstupných rozhraniach však priepustnosť klesá pre všetky merané konfigurácie na 76% vyťaženia linky. K tomuto javu dochádza z dôvodu generovania zhlukov malých paketov prístrojom Spirent. Aj keď priemerná veľkosť rámcov je 700 bajtov počas veľmi krátkych okamžikov kriticky rastie počet generovaných paketov, čo spôsobuje zahadzovanie na vstupných rozhraniach sondy.



Obr. 9.6: Graf priepustnosti pre 10000 tokov

Pri pozornom skúmaní grafu na obrázku 9.6 si môžeme všimnúť rozdiely v priepustnosti medzi konfiguráciami s hlavičkami protokolu IPv4 a IPv6 . Konkrétne pri meraniach so 64 a 78 bajtovými rámcami. Keďže v predchádzajúcom grafe sa podobné rozdiely nevyskytovali, je zrejmé že sa táto vlastnosť prejavuje iba pri väčšom počte tokov. Tento 1-2 % rozdiel v

maximálnej priepustnosti, môže byť spôsobený väčšou réžiou pri exporte záznamov o IPv6 tokoch. Tieto záznamy sú o proti IPv4 záznamom o takmer 20 % väčšie, čo môže predstavovať nezanedbateľný nárast réžie, najmä v extrémnych prípadoch, kedy sú všetky toky exportované v jeden okamžik, čo sa dej aj pri týchto meraniach.

Okrem toho, pri veľkom počte tokov je réžia exportu veľmi výrazná. Prejavuje sa zahadzovaním paketov na vstupnom rozhraní v pravidelných intervaloch odpovedajúcim dĺžke aktívneho prípadne neaktívneho time-outu. Počas meraní priepustnosti som zistil, že tento jav znižuje priepustnosť, pri veľkom počte tokov, až o 5 % vyťaženia linky.

Kapitola 10

Záver

Cieľom tejto práce bolo navrhnúť a implementovať systém pre monitorovanie 10 Gb sietí, ktorý poskytuje informácie o štruktúre sieťovej prevádzky vo forme záznamov o tokoch a stromu protokolov. Navyše umožňuje zachytiť pakety na základe sekvencie zapúzdrenia protokolov a všetky získané dáta perzistentne ukladať. Ďalej bolo potrebné implementovať grafické užívateľské rozhranie a celý systém otestovať. Všetky stanovené ciele boli splnené.

Návrhu a implementácii predchádzala príprava, počas ktorej bolo potrebné naštudovať problematiku monitorovania sietí so zameraním sa na monitoring tokov a štruktúru sieťovej prevádzky.

Na základe naštudovaných poznatkov som identifikoval informácie, ktoré je nutné z paketov extrahovať. Aby bolo možné tieto informácie získavať aj vo vysokorýchlostných sieťach, zameral som sa na analýzy a extrakcie polí z hlavičiek protokolov v technológii FPGA. S cieľom dosiahnuť flexibilitu vyplývajúcu z možnosti rekonfigurovať hradlové polia, som sa oboznámil s vysoko-úrovňovým jazykom P4, ktorý umožňuje popisovať náväznosti protokolov v rámci paketu. Použil som vlastný generátor, ktorý vznikol ako súčasť mojej bakalárskej práce [32]. Tento program prevádza popis procesu parsovania v jazyku P4, na popis komponenty analyzujúcej hlavičky protokolov v jazyku VHDL. Po niekoľkých zmenách v nástroji vygenerované komponenty poskytujú všetky informácie, ktoré potrebujeme a navyše dosahujú priepustnosť 10 Gbps.

Ďalej som navrhnol monitorovací systém, dekomponoval ho na podúlohy a tie mapoval na výpočtové prostriedky platformy, ktorá bola pre tento monitorovací systém navrhnutá. Na základe návrhu popísaného v kapitole 6 bola implementovaná softvérová aj hardvérová časť sondy a vykonané úpravy v kóde kolektoru. Ďalej som vytvoril grafické užívateľské rozhranie, umožňujúce konfiguráciu hardvérových komponent na sonde. Potom som dôkladne overil správnosť implementácie pomocou vlastného automatizovaného testovacieho prostredia a funkčných verifikácí.

Za vynikajúci, považujem výsledok meraní spotreby zdrojov v FPGA, ktorá je v prípade ALM 8 % a u ostatných zdrojov dokonca menej ako 1 %. Z dôvodu nedostupnosti cieľovej platformy boli merania priepustnosti vykonané na pomalšej platforme s dvoma 1 Gb vstupnými rozhraniami. Napriek tomu celková priepustnosť, už pri paketoch veľkých 256 bajtov, dosahuje 2 Gbps, teda plnú linkovú rýchlosť z oboch vstupov.

V budúcnosti je možné monitorovací systém rozšíriť o ďalšiu funkcionalitu. Jednou z možností je získavanie sekvenčných čísel a príznakov z TCP tokov. Pomocou nich môžeme detekovať retransmisie alebo straty paketov a tak hodnotiť úplnosť tokov na sieti.

Literatúra

[1] Arm Corporated: *ARM® Cortex®-A72 MPCore Processor Technical Reference Manual*. [online; navštívené 20.4.2018].
URL http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.100095_0001_02_en/Chunk214275741.html

[2] Benáček, P.; Puš, V.; Kubátová, H.; aj.: P4-To-VHDL: Automatic generation of high-speed input and output network blocks. *Microprocessors and Microsystems*, ročník 56, č. Supplement C, 2018: s. 22 – 33, ISSN 0141-9331, doi:https://doi.org/10.1016/j.micpro.2017.10.012.
URL http://www.sciencedirect.com/science/article/pii/S0141933117304787

[3] Bosshart, P.; Daly, D.; Gibb, G.; aj.: *P4: Programming Protocol-Independent Packet Processors*. Computer Communication Review, [Online; navštívené 8.1.2018].
URL www.sigcomm.org/sites/default/files/ccr/papers/2014/July/0000000-0000004.pdf

[4] Brownlee, N.; Mills, C.; Ruth, G.: Traffic Flow Measurement: Architecture. RFC 2722, RFC Editor, October 1999.
URL https://tools.ietf.org/html/rfc2722

[5] Case, J.; Fedor, M.; Schoffstall, M.; aj.: A Simple Network Management Protocol. RFC 1067, RFC Editor, August 1988.
URL https://tools.ietf.org/html/rfc1067

[6] Cesnet: IPFIXcol. 2014, [Online; navštívené 26.04.2018.
URL https://github.com/CESNET/ipfixcol/blob/master/base/

[7] Cesnet: IPFIXcol architecture scheme. 2014, [Online; navštívené 26.04.2018.
URL https://github.com/CESNET/ipfixcol/blob/master/base/documentation/img/arch_scheme.png

[8] Cisco Systems Inc.: *Introduction to Cisco IOS NetFlow - A Technical Overview*. [Online; navštívené 05.12.2017].
URL https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html

[9] Claise, B.: Cisco Systems NetFlow Services Export Version 9. RFC 3954, RFC Editor, October 2004.
URL https://tools.ietf.org/html/rfc3954

[10] Claise, B.: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101, RFC Editor, January 2008.
URL https://tools.ietf.org/html/rfc5101

[11] Claise, B.; Trammell, B.; Aitken, P.: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 7011, RFC Editor, September 2013.
URL https://tools.ietf.org/html/rfc7011

[12] Funiak, M.: *Hardwarově akcelerovaná funkční verifikace procesoru, bakalářská práce, Brno, FIT VUT v Brně, 2013.*

[13] Gibb, G.; Varghese, G.; Horowitz, M.; aj.: *Design Principles for Packet Parsers.* ANCS'13, *ACM/IEEE Symposium, [online], Vytvorené v októbri 2013.*
URL http://yuba.stanford.edu/~nickm/papers/ancs48-gibb.pdf

[14] Hofstede, R.; Celeda, P.; Trammell, B.; aj.: Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX. IEEE, vol. 16, no. 4 2014, [Online; navštívené 12.12.2017].
URL http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6814316

[15] Inc., C. S.: Cisco IOS Software Release 11.1. 2006, [Online; navštívené 5.1.2018.
URL https://www.cisco.com/en/US/products/sw/iosswrel/ps1820/products_tech_note09186a0080094482.shtml

[16] Intel Corporation: Intel® Arria® 10 Device Overview. [online; navštívené 2.1.2018].
URL http://www.mouser.com/ds/2/612/a10_overview-1098925.pdf

[17] Intel Corporation: Intel® Arria® 10 FPGA Performance Benchmarking Methodology and Results. [online; navštívené 2.1.2018].
URL https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01271-intel-arria10-performance-benchmarking-methodology-and-results.pdf

[18] Internet Assigned Numbers Authority: IP Flow Information Export (IPFIX) Entities. [Online; navštíveno 28.11.2017].
URL https://www.iana.org/assignments/ipfix/ipfix.xhtml

[19] Klaška, L.: SNMP objekty a MIB. 2000, [Online; navštívené 3.1.2018.
URL http://www.svetsiti.cz/tutorialy/mngmt/pruvodce/obr8.gif

[20] Kozanitis, C.; Huber, J.; Singh, S.; aj.: Leaping multiple headers in a single bound: wire-speed parsing using the Kangaroo system. INFOCOM 2010, [online], Vytvorené v marci 2010.
URL http://cseweb.ucsd.edu/~ckozanit/files/infocom2010.pdf

[21] Kozubík, M.: Profilování dat pomocí IPFIX mediátoru, bakalářská práce, Brno, FIT VUT v Brně, 2015.

[22] Košař, V.: Verifikace systemu pro detekci nežadoucího provozu, bakalařska prace, Brno, FIT VUT v Brně, 2008.

[23] *Kretchmar, J. M.; Dostálek, L.: Administrace a diagnostika sítí: pomocí OpenSource utilit a nástrojů. Computer Press, 2004, ISBN 8025103455.*

[24] *Netcope Technologies a.s.: Netcope P4. [Online; navštívené 09.1.2018].*
URL
https://www.xilinx.com/products/intellectual-property/1-pcz517.html

[25] *NetFlow: NetFlow — Wikipedia, The Free Encyclopedia. 2017, [Online; navštívené 30.11.2017].*
URL https://en.wikipedia.org/wiki/NetFlow

[26] *Orebaugh, A.; Ramirez, G.; Beale, J.:* Wireshark & Ethereal Network Protocol Analyzer Toolkit*. Jay Beale's open source security series, Elsevier Science, 2006, ISBN 9780080506012.*

[27] *Pepelnjak, I.: Management, Control and Data Planes in Network Devices and Systems. 2013, [Online; navštívené 9.12.2017.*
URL http://4.bp.blogspot.com/-LNsgPsARtHU/UgMqsuYa9lI/AAAAAAAAGCA/awbkU1X9nLU/s1600/DevicePlanes.jpg

[28] *Puš, V.; Kořenek, J.: Design Methodology of Configurable High Performance Packet Parser for FPGA. Design and Diagnostics of Electronic Circuits & Systems , [online], Vytvorené 2014.*
URL https://www.liberouter.org/wp-content/uploads/2013/02/hfe.pdf

[29] *Rose, M.; McCloghrie, K.: Structure and Identification of Management Information for TCP/IP-based internets. RFC 1065, RFC Editor, August 1988.*
URL https://tools.ietf.org/html/rfc1065

[30] *Rose, M.; McCloghrie, K.: Structure and Identification of Management Information for TCP/IP-based Internets. RFC 1155, RFC Editor, May 1990.*
URL https://tools.ietf.org/html/rfc1155

[31] *Sanders, C.:* Practical packet analysis: using wireshark to solve real-world network problems*. No starch press, 2011, ISBN 9781593272661.*

[32] *Selecký, R.: Analýza a extrakce položek z hlaviček paketů v FPGA, bakalářská práce, Brno, FIT VUT v Brně, 2016.*

[33] *The P4 Language Consortium: P4-HLIR. GitHub Inc., [online; navštívené 19.12.2017].*
URL https://github.com/p4lang/p4-hlir

[34] *The P4 Language Consortium: The $P4_{16}$ Language Specification, version 1.0.0. May "2017", [Online; navštívené 20.12.2017].*
URL https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.pdf

[35] *Trammell, B.; Boschi, E.: An introduction to IP flow information export (IPFIX).* IEEE Communications Magazine*, ročník 49, č. 4, April 2011: s. 89–95, ISSN 0163-6804, doi:10.1109/MCOM.2011.5741152.*

[36] *Waters, G.: User-based Security Model for SNMPv2. RFC 1910, RFC Editor, February 1996.*
URL https://tools.ietf.org/html/rfc1910

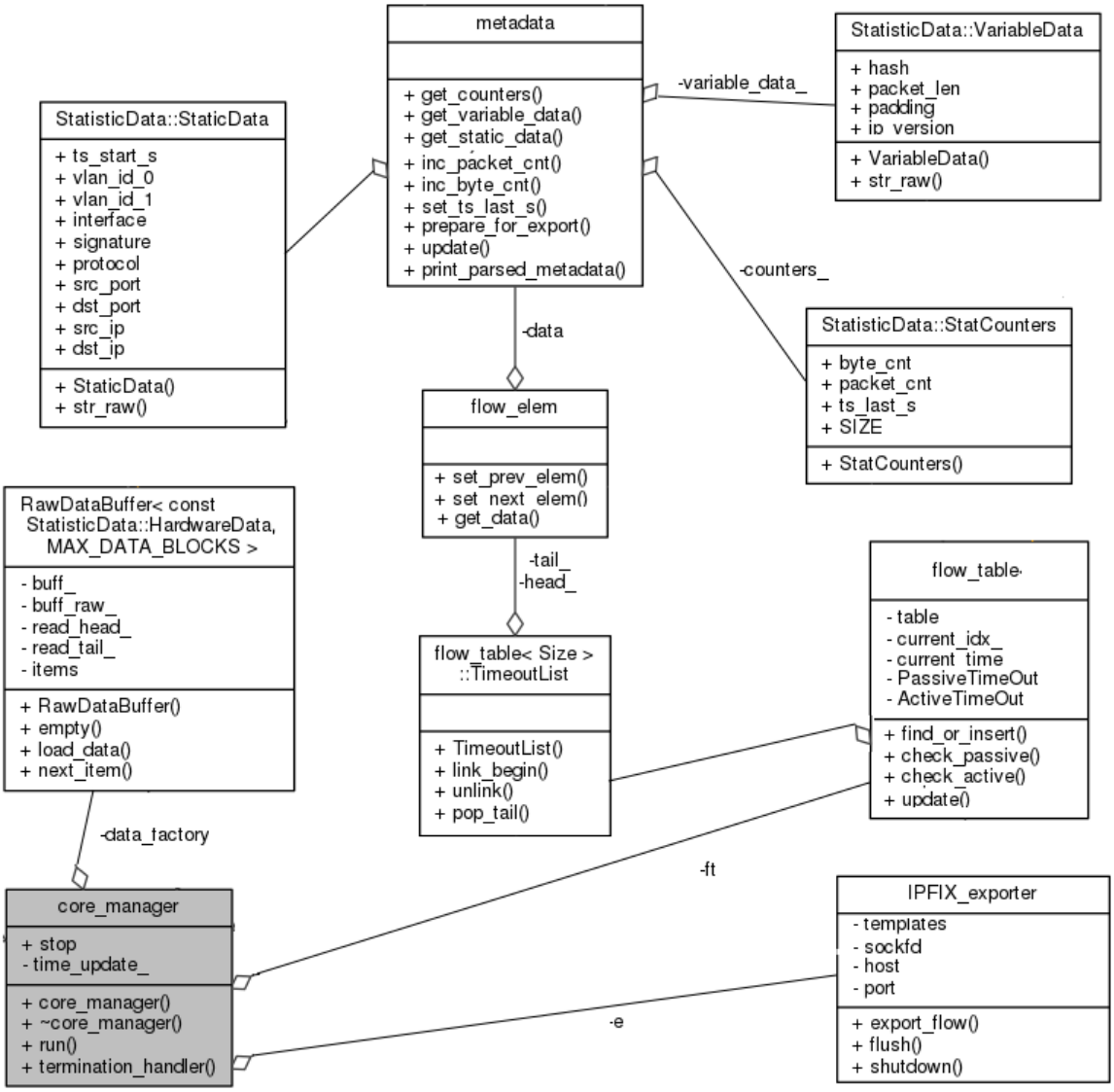
[37] *WWW stránky: Intel Corporation, Arria 10. [online; navštívené 2.1.2018].*
URL https://www.altera.com/products/fpga/arria-series/arria-10/overview.html

[38] *WWW stránky: NXP Semiconductors, QorIQ®. [online; navštívené 22.12.2017].*
URL http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/qoriq-layerscape-arm-processors/qoriq-layerscape-2088a-and-2048a-multicore-communications-processors:LS2088A

[39] *Šoltés, M.: Nástroj pro práci s daty NetFlow, bakalářská práce, Brno, FIT VUT v Brně, 2011.*

Príloha A

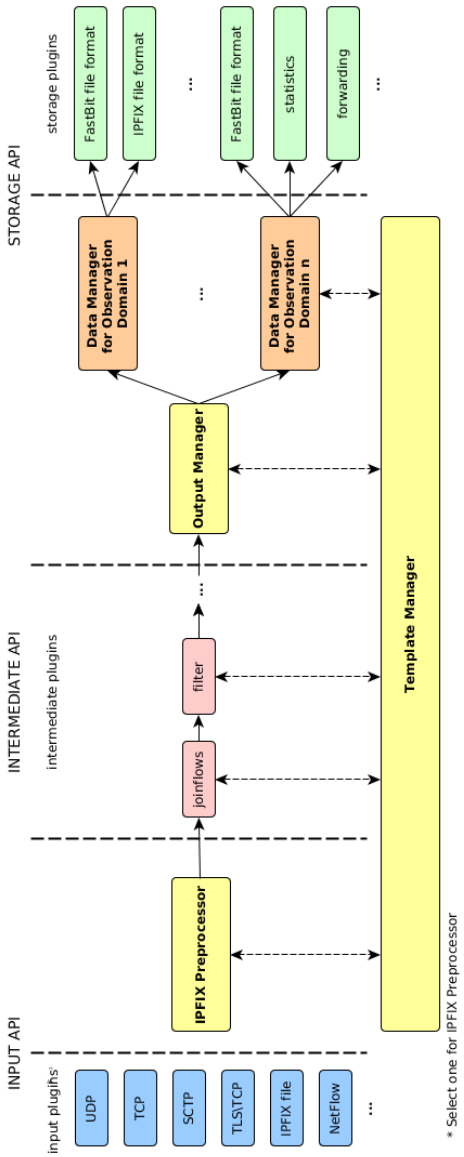
Diagram tried



Obr. A.1: Diagram tried softwárovej časti monitorovacieho systému

Príloha B

Architektúra IPFIXcol



Obr. B.1: Schéma architektúry IPFIXcol [7]

Príloha C

Namerané hodnoty priepustnosti

	1 wire				2 wires			
	1 flow		10000 flows		1 flow		10000 flows	
frame size	throughput		throughput		throughput		throughput	
[B]	[Bps]	[%]	[Bps]	[%]	[Bps]	[%]	[Bps]	[%]
64	95238116	100	69523805	73	99047600	104	59047800	62
78	99489816	100	85561232	86	121377594	122	79591836	80
128	108108129	100	108108129	100	198918948	184	129729756	120
256	115942073	100	115942073	100	231884146	200	231884146	200
512	120300649	100	120300649	100	240601298	200	240601298	200
∼ 700	121525900	100	121525900	100	184663430	152	184663430	152
1024	122605378	100	122605378	100	2452107567	200	245210756	200
1500	123355194	100	123355194	100	246710388	200	246710388	200

Tabuľka C.1: Priepustnosť vybraných konfigurácií s protokolom IPv4

	1 wire				2 wires			
	1 flow		10000 flows		1 flow		10000 flows	
frame size	throughput		throughput		throughput		throughput	
[B]	[Bps]	[%]	[Bps]	[%]	[Bps]	[%]	[Bps]	[%]
64	-	-	-	-	-	-	-	-
78	99489782	100	84556337	85	119387732	120	75612280	76
128	108108129	100	108108129	100	198918948	184	129729756	120
256	115942073	100	115942073	100	231884146	200	231884146	200
512	120300649	100	1203006497	100	240601298	200	240601298	200
∼ 700	121525900	100	121525900	100	184701568	152	184690000	152
1024	122605378	100	122605378	100	245210756	200	245210756	200
1500	123355194	100	123355194	100	246710388	200	246710388	200

Tabuľka C.2: Priepustnosť vybraných konfigurácií s protokolom IPv6

frame size	1 flow		100 flows		10000 flows		100000 flows	
	throughput		throughput		throughput		throughput	
[B]	[fps]	[%]	[fps]	[%]	[fps]	[%]	[fps]	[%]
64	1488095	100	1413690	95	1086310	73	818452	55
128	844595	100	844595	100	844595	100	844595	100
256	452898	100	452898	100	452898	100	452898	100
512	234963	100	234963	100	234963	100	234963	100
∼ 700	173647	100	173647	100	173647	100	173647	100
1024	119732	100	119732	100	119732	100	119732	100
1500	82237	100	82237	100	82237	100	82237	100

Tabuľka C.3: Priepustnosť konfigurácií s 1 vstupným rozhraním a protokolom IPv4

frame size	1 flow		100 flows		10000 flows		100000 flows	
	throughput		throughput		throughput		throughput	
[B]	[fps]	[%]	[fps]	[%]	[fps]	[%]	[fps]	[%]
64	773809	52	669643	45	461096	31	372024	25
128	777027	92	675676	80	506757	60	371621	44
256	452898	100	452898	100	452898	100	384964	85
512	234963	100	234963	100	234963	100	234963	100
∼ 700	132200	76	133998	77	132200	76	133998	77
1024	119732	100	119732	100	119732	100	119732	100
1500	82237	100	82237	100	82237	100	82237	100

Tabuľka C.4: Priepustnosť konfigurácií s 2 vstupnými rozhraniami a protokolom IPv4

frame size	1 flow		100 flows		10000 flows		100000 flows	
	throughput		throughput		throughput		throughput	
[B]	[fps]	[%]	[fps]	[%]	[fps]	[%]	[fps]	[%]
78	1275511	100	1275511	100	1084184	85	918367	72
128	844595	100	844595	100	844595	100	844595	100
256	452898	100	452898	100	452898	100	452898	100
512	234963	100	234963	100	234963	100	234963	100
∼ 700	173647	100	173647	100	173647	100	173647	100
1024	119732	100	119732	100	119732	100	119732	100
1500	82237	100	82237	100	82237	100	82237	100

Tabuľka C.5: Priepustnosť konfigurácií s 1 vstupným rozhraním a protokolom IPv6

frame size	1 flow		100 flows		10000 flows		100000 flows	
	throughput		throughput		throughput		throughput	
[B]	[fps]	[%]	[fps]	[%]	[fps]	[%]	[fps]	[%]
78	765306	60	669643	52	484694	38	369898	29
128	777027	92	667230	79	506757	60	380068	40
256	452898	100	452898	100	452898	100	371377	82
512	234963	100	234963	100	234963	100	234963	100
$\sim$ 700	132019	76	132019	76	132007	76	132019	76
1024	119732	100	119732	100	119732	100	119732	100
1500	82237	100	82237	100	82237	100	82237	100

Tabuľka C.6: Priepustnosť konfigurácií s 2 vstupnými rozhraniami a protokolom IPv6

Príloha D

Obsah priloženého pamäťového média

Priložené CD obsahuje:

- priečinok **sw** obsahujúci zdrojové súbory softvérovej časti
- priečinok **hw** obsahujúci zdrojové súbory hardvérovej časti
- priečinok **gui** obsahujúci zdrojové súbory grafického užívateľského rozhrania
- priečinok **ipfixcol** obsahujúci konfiguračné a zdrojové súbory programu IPFIXcol
- priečinok **ver** obsahujúci zdrojové súbory verifikačného prostredia
- priečinok **scapy_tester** obsahujúci súbory automatizovaného testovacieho prostredia
- priečinok **doc** obsahujúci obrázky a zdrojové súbory diplomovej práce pre LaTeX